

①

AD-A153 607

INTERNET PROTOCOL TRANSITION WORKBOOK

March 1982

DTIC
ELECTE
MAY 13 1985
B

DTIC FILE COPY



Network Information Center
SRI International
Menlo Park, CA 94025
(NIC@NIC)

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION U			1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY			3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release. Distribution unlimited.		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE			4. MONITORING ORGANIZATION REPORT NUMBER(S)		
4. PERFORMING ORGANIZATION REPORT NUMBER(S)			5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION Network Information Center SRI International		6b. OFFICE SYMBOL (if applicable)	7a. NAME OF MONITORING ORGANIZATION Defense Data Network- PMO		
6c. ADDRESS (City, State, and ZIP Code) Menlo Park, CA 94025			7b. ADDRESS (City, State, and ZIP Code) Washington, DC		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION		8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER YCA200-83-C-0025		
8c. ADDRESS (City, State, and ZIP Code)			10. SOURCE OF FUNDING NUMBERS		
			PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
			WORK UNIT ACCESSION NO.		
11. TITLE (include Security Classification) Internet Protocol Transition Workbook					
12. PERSONAL AUTHOR(S)					
13a. TYPE OF REPORT		13b. TIME COVERED FROM TO		14. DATE OF REPORT (Year, Month, Day) 520300	
				15. PAGE COUNT 521	
16. SUPPLEMENTARY NOTATION					
17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB-GROUP	1p; Internet Protocol; TCP; Transmission Control Protocol; FTP; File Transfer Protocol; Catenet; Datagram; Mail header;		
19. ABSTRACT (Continue on reverse if necessary and identify by block number) This document contains the Internet Protocols, including the Transmission Control Protocol, Internet Protocol, Telnet Protocol, Mail Protocol, and File Transfer Protocols. Numerous other standards and RFCs useful for bringing a host onto the network are provided, as is the NCP/TCP Transition Plan, which is interesting from a historical standpoint. Also included is general information on implementing these protocols, and information on related topics such as assigned numbers.					
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT <input checked="" type="checkbox"/> UNCLASSIFIED/UNLIMITED <input type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS			21. ABSTRACT SECURITY CLASSIFICATION Unclassified		
22a. NAME OF PERSONS OR ORGANIZATION			22b. TELEPHONE (include Area Code)		22c. OFFICE SYMBOL

INTERNET PROTOCOL TRANSITION WORKBOOK

March 1982



**Network Information Center
SRI International
Menlo Park, CA 94025
(NIC@NIC)**

TABLE OF CONTENTS

CONTACTS	11
INTRODUCTION	
THE NCP/TCP TRANSITION PLAN (RFC-801)	1
CATENET MODEL (IEN-48)	25
NETWORK LEVEL	
INTERNET PROTOCOL (RFC-791)	47
INTERNET CONTROL MESSAGE PROTOCOL (RFC-792)	101
HOST LEVEL	
USER DATAGRAM PROTOCOL (RFC-768)	125
TRANSMISSION CONTROL PROTOCOL (RFC-793)	131
APPLICATION LEVEL	
TELNET PROTOCOL (RFC-764)	223
FILE TRANSFER PROTOCOL (RFC-765)	241
SIMPLE MAIL TRANSFER PROTOCOL (RFC-788)	313
TRIVIAL FILE TRANSFER PROTOCOL (RFC-783)	379
NAME SERVER PROTOCOL (IEN-116)	397
APPENDICES	
ASSIGNED NUMBERS (RFC-790)	409
PRE-EMPTION (RFC-794)	427
SERVICE MAPPINGS (RFC-795)	431
ADDRESS MAPPINGS (RFC-796)	437
MAIL HEADER FORMAT STANDARDS (RFC-733)	447
C/30 INFORMATION	493
HOST TABLE SPECIFICATION (RFC-810)	505
HOSTNAME SERVER (RFC-811)	513
NICNAME/WHOIS SERVER (RFC-812)	517

Note: page numbers listed in the Table of Contents refer to the page numbers in parentheses. Other paging pertains only to the protocol in which it occurs. This paging has been left intact so that page references within the text of a given protocol that refer to the original paging will still have meaning.

March 1982

Internet Protocol
Transition Workbook

CONTACTS

ARPANET MANAGEMENT, POLICY, AND SERVICE

Questions regarding general policy or access to the ARPANET or unsatisfactory ARPANET service should be directed to:

Major Glynn Parker
Director, Defense Communications Agency (DCA)
ARPANET Management Branch,
Code252
Washington, D. C. 20305

(202) 692-6175 (Commercial)
222-6175 (Autovon)
DCACODE252@ISI (Online mail)

INTERNET PROTOCOLS

Technical questions regarding transition to internet protocols should be directed to:

Dr. Jon Postel
University of Southern California
Information Science Institute
4676 Admiralty Way
Marina Del Rey, California 90291

(213) 822-1511
POSTEL@ISIF (Online mail)

GENERAL NETWORK PROTOCOL INFORMATION

Questions regarding where protocols are located on the network or who to contact should be directed to:

Elizabeth Feinler
Network Information Center
SRI International
Menlo Park, California 94025

(415) 859-3695
FEINLER@SRI-NIC (Online mail)

Technical questions concerning C/30 IMPs or TACs should be directed to:

(617) 497-3117
HERMAN@BBN-UNIX (Online mail)



Application for	
Project No.	✓
Date Recd.	
Name of Applicant	
Address	
Phone Number	
Remarks	
Approved by _____	
District _____	
Signature _____	
Date _____	
A-1	

INTRODUCTION

THE NCP/TCP TRANSITION PLAN

THE CATENET MODEL FOR INTERNETWORKING

RFC-801

THE NCP/TCP TRANSITION PLAN

September 1981

NCP/TCP TRANSITION PLAN

Introduction

ARPA sponsored research on computer networks led to the development of the ARPANET. The installation of the ARPANET began in September 1969, and regular operational use was underway by 1971. The ARPANET has been an operational service for at least 10 years. Even while it has provided a reliable service in support of a variety of computer research activities, it has itself been a subject of continuing research, and has evolved significantly during that time.

In the past several years ARPA has sponsored additional research on computer networks, principally networks based on different underlying communication techniques, in particular, digital packet broadcast radio and satellite networks. Also, in the ARPA community there has been significant work on local networks.

It was clear from the start of this research on other networks that the base host-to-host protocol used in the ARPANET was inadequate for use in these networks. In 1973 work was initiated on a host-to-host protocol for use across all these networks. The result of this long effort is the Internet Protocol (IP) and the Transmission Control Protocol (TCP).

These protocols allow all hosts in the interconnected set of these networks to share a common interprocess communication environment. The collection of interconnected networks is called the ARPA Internet (sometimes called the "Catenet").

The Department of Defense has recently adopted the internet concept and the IP and TCP protocols in particular as DoD wide standards for all DoD packet networks, and will be transitioning to this architecture over the next several years. All new DoD packet networks will be using these protocols exclusively.

The time has come to put these protocols into use in the operational ARPANET, and extend the logical connectivity of the ARPANET hosts to include hosts in other networks participating in the ARPA Internet.

As with all new systems, there will be some aspects which are not as robust and efficient as we would like (just as with the initial ARPANET). But with your help, these problems can be solved and we

can move into an environment with significantly broader communication services.

Discussion

The implementation of IP/TCP on several hosts has already been completed, and the use of some services is underway. It is urgent that the implementation of IP/TCP be begun on all other ARPANET hosts as soon as possible and no later than 1 January 1982 in any case. Any new host connected to the ARPANET should only implement IP/TCP and TCP-based services. Several important implementation issues are discussed in the last section of this memo.

Because all hosts can not be converted to TCP simultaneously, and some will implement only IP/TCP, it will be necessary to provide temporarily for communication between NCP-only hosts and TCP-only hosts. To do this certain hosts which implement both NCP and IP/TCP will be designated as relay hosts. These relay hosts will support Telnet, FTP, and Mail services on both NCP and TCP. These relay services will be provided beginning in November 1981, and will be fully in place in January 1982.

Initially there will be many NCP-only hosts and a few TCP-only hosts, and the load on the relay hosts will be relatively light. As time goes by, and the conversion progresses, there will be more TCP capable hosts, and fewer NCP-only hosts, plus new TCP-only hosts. But, presumably most hosts that are now NCP-only will implement IP/TCP in addition to their NCP and become "dual protocol" hosts. So, while the load on the relay hosts will rise, it will not be a substantial portion of the total traffic.

The next section expands on this plan, and the following section gives some milestones in the transition process. The last section lists the key documents describing the new protocols and services. Appendices present scenarios for use of the relay services.

The General Plan

The goal is to make a complete switch over from the NCP to IP/TCP by 1 January 1983.

It is the task of each host organization to implement IP/TCP for its own hosts. This implementation task must begin by 1 January 1982.

IP:

This is specified in RFCs 791 and 792. Implementations exist for several machines and operating systems. (See Appendix D.)

TCP:

This is specified in RFC793. Implementations exist for several machines and operating systems. (See Appendix D.)

It is not enough to implement the IP/TCP protocols, the principal services must be available on this IP/TCP base as well. The principal services are: Telnet, File Transfer, and Mail.

It is the task of each host organization to implement the principal services for its own hosts. These implementation tasks must begin by 1 January 1982.

Telnet:

This is specified in RFC 764. It is very similar to the Telnet used with the NCP. The primary differences are that the ICP is eliminated, and the NCP Interrupt is replaced with the TCP Urgent.

FTP:

This is specified in RFC 765. It is very similar to the FTP used with the NCP. The primary differences are that in addition to the changes for Telnet, that the data channel is limited to 8-bit bytes so FTP features to use other transmission byte sizes are eliminated.

Mail:

This is specified in RFC 788. Mail is separated completely from FTP and handled by a distinct server. The procedure is similar in concept to the old FTP/NCP mail procedure, but is very different in detail, and supports additional functions especially mail relaying, and multi-recipient delivery.

Beyond providing the principal services in the new environment, there must be provision for interworking between the new environment and the old environment between now and January 1983.

For Telnet, there will be provided one or more relay hosts. A Telnet relay host will implement both the NCP and TCP environments and both user and server Telnet in both environments. Users requiring Telnet service between hosts in different environments

will first connect to a Telnet relay host and then connect to the destination host. (See Appendix A.)

For FTP, there will be provided one or more relay hosts. An FTP relay host will implement both the NCP and TCP environments, both user and server Telnet, and both user and server FTP in both environments. Users requiring FTP service between hosts in different environments will first connect via Telnet to an FTP relay host, then use FTP to move the file from the file donor host to the FTP relay host, and finally use FTP to move the file from the FTP relay host to the file acceptor host. (See Appendix B.)

For Mail, hosts will implement the new Simple Mail Transfer Protocol (SMTP) described in RFC 788. The SMTP procedure provides for relaying mail among several protocol environments. For TCP-only hosts, using SMTP will be sufficient. For NCP-only hosts that have not been modified to use SMTP, the special syntax "user.host@forwarder" may be used to relay mail via one or more special forwarding host. Several mail relay hosts will relay mail via SMTP procedures between the NCP and TCP environments, and at least one special forwarding host will be provided. (See Appendix C.)

Milestones

First Internet Service	already
A few hosts are TCP-capable and use TCP-based services.	
First TCP-only Host	already
The first TCP-only host begins use of TCP-based services.	
Telnet and FTP Relay Service	already
Special relay accounts are available to qualified users with a demonstrated need for the Telnet or FTP relay service.	
Ad Hoc Mail Relay Service	already
An ad hoc mail relay service using the prototype MTP (RFC 780) is implemented and mail is relayed from the TCP-only hosts to NCP-only hosts, but not vice versa. This service will be replaced by the SMTP service.	
Last NCP Conversion Begins	Jan 82
The last NCP-only host begins conversion to TCP.	

Mail Relay Service

Jan 82

The SMTP (RFC 788) mail service begins to operate and at least one mail relay host is operational, and at least one special forwarder is operational to provide NCP-only host to TCP-only host mail connectivity.

Normal Internet Service

Jul 82

Most hosts are TCP-capable and use TCP-based services.

Last NCP Conversion Completed

Nov 82

The last NCP-only host completes conversion to TCP.

Full Internet Service

Jan 83

All hosts are TCP-capable and use TCP-based services. NCP is removed from service, relay services end, all services are TCP-based.

Documents

The following RFCs document the protocols to be implemented in the new IP/TCP environment:

IP	RFC 791
ICMP	RFC 792
TCP	RFC 793
Telnet	RFC 764
FTP	RFC 765
SMTP	RFC 788
Name Server	IES 116
Assigned Numbers	RFC 790

These and associated documents are to be published in a notebook, and other information useful to implementers is to be gathered. These documents will be made available on the following schedule:

Internet Protocol Handbook	Jan 82
Implementers Hints	Jan 82
SDC IP/TCP Specifications	Jan 82
Expanded Host Table	Jan 82

Implementation Issues

There are several implementation issues that need attention, and there are some associated facilities with these protocols that are not necessarily obvious. Some of these may need to be upgraded or redesigned to work with the new protocols.

Name Tables

Most hosts have a table for converting character string names of hosts to numeric addresses. There are two effects of this transition that may impact a host's table of host names: (1) there will be many more names, and (2) there may be a need to note the protocol capability of each host (SMTP/TCP, SMTP/NCP, FTP/NCP, etc.).

Some hosts have kept this table in the operating system address space to provide for fast translation using a system call. This may not be practical in the future.

There may be applications that could take alternate actions if they could easily determine if a remote host supported a particular protocol. It might be useful to extend host name tables to note which protocols are supported.

It might be necessary for the host name table to contain names of hosts reachable only via relays if this name table is used to verify the spelling of host names in application programs such as mail composition programs.

It might be advantageous to do away with the host name table and use a Name Server instead, or to keep a relatively small table as a cache of recently used host names.

A format, distribution, and update procedure for the expanded host table will be published soon.

Mail Programs

It may be possible to move to the new SMTP mail procedures by changing only the mailer-daemon and implementing the SMTP-server, but in some hosts there may be a need to make some small changes to some or all of the mail composition programs.

There may be a need to allow users to identify relay hosts for messages they send. This may require a new command or address syntax not now currently allowed.

IP/TCP

Continuing use of IP and TCP will lead to a better understanding of the performance characteristics and parameters. Implementers should expect to make small changes from time to time to improve performance.

Shortcuts

There are some very tempting shortcuts in the implementation of IP and TCP. DO NOT BE TEMPTED! Others have and they have been caught! Some deficiencies with past implementations that must be remedied and are not allowed in the future are the following:

IP problems:

Some IP implementations did not verify the IP header checksum.

Some IP implementations did not implement fragment reassembly.

Some IP implementations used static and limited routing information, and did not make use of the ICMP redirect message information.

Some IP implementations did not process options.

Some IP implementations did not report errors they detected in a useful way.

TCP problems:

Some TCP implementations did not verify the TCP checksum.

Some TCP implementations did not reorder segments.

Some TCP implementations did not protect against silly window syndrome.

Some TCP implementations did not report errors they detected in a useful way.

Some TCP implementations did not process options.

Host problems:

Some hosts had limited or static name tables.

Relay Service

The provision of relay services has started. There are two concerns about the relay service: (1) reliability, and (2) load.

The reliability is a concern because relaying puts another host in the chain of things that have to all work at the same time to get the job done. It is desirable to provide alternate relay hosts if possible. This seems quite feasible for mail, but it may be a bit sticky for Telnet and FTP due to the need for access control of the login accounts.

The load is a potential problem, since an overloaded relay host will lead to unhappy users. This is another reason to provide a number of relay hosts, to divide the load and provide better service.

A Digression on the Numbers

How bad could it be, this relay load? Essentially any "dual protocol" host takes itself out of the game (i.e., does not need relay services). Let us postulate that the number of NCP-only hosts times the number of TCP-only hosts is a measure of the relay load.

Total Hosts	Dual Hosts	NCP Hosts	TCP Hosts	"Load"	Date
200	20	178	2	356	Jan-82
210	40	158	12	1896	Mar-82
220	60	135	25	3375	May-82
225	95	90	40	3600	Jul-82
230	100	85	45	3825	Sep-82
240	125	55	60	3300	Nov-82
245	155	20	70	1400	Dec-82
250	170	0	80	0	31-Dec-82
250	0	0	250	0	1-Jan-83

This assumes that most NCP-only hosts (but not all) will become to dual protocol hosts, and that 50 new host will show up over the course of the year, and all the new hosts are TCP-only.

If the initial 200 hosts immediately split into 100 NCP-only and 100 TCP-only then the "load" would be 10,000, so the fact that most of the hosts will be dual protocol hosts helps considerably.

This load measure (NCP hosts times TCP hosts) may over state the load significantly.

Please note that this digression is rather speculative!

Gateways

There must be continuing development of the internet gateways.
The following items need attention:

- Congestion Control via ICMP

- Gateways use connected networks intelligently

- Gateways have adequate buffers

- Gateways have fault isolation instrumentation

Note that the work in progress on the existing gateways will provide the capability to deal with many of these issues early in 1982. Work is also underway to provide improved capability gateways based on new hardware late in 1982.

APPENDIX A. Telnet Relay Scenario

Suppose a user at a TCP-only host wishes to use the interactive services of an NCP-only service host.

- 1) Use the local user Telnet program to connect via Telnet/TCP to the RELAY host.
- 2) Login on the RELAY host using a special account for the relay service.
- 3) Use the user Telnet on the RELAY host to connect via Telnet/NCP to the service host. Since both Telnet/TCP and Telnet/NCP are available on the RELAY host the user must select which is to be used in this step.
- 4) Login on the service host using the regular account.



Suppose a user at a NCP-only host wishes to use the interactive services of an TCP-only service host.

- 1) Use the local user Telnet program to connect via Telnet/NCP to the RELAY host.
- 2) Login on the RELAY host using a special account for the relay service.
- 3) Use the user Telnet on the RELAY host to connect via Telnet/NCP to the service host. Since both Telnet/TCP and Telnet/NCP are available on the RELAY host the user must select which is to be used in this step.
- 4) Login on the service host using the regular account.



APPENDIX B. FTP Relay Scenario

Suppose a user at a TCP-only host wishes copy a file from a NCP-only donor host.

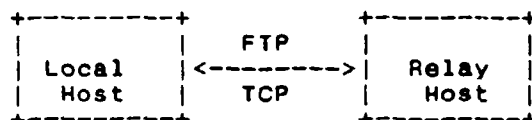
Phase 1:

- 1) Use the local user Telnet program to connect via Telnet/TCP to the RELAY host.
- 2) Login on the RELAY host using a special account for the relay service.
- 3) Use the user FTP on the RELAY host to connect via FTP/NCP to the donor host.
- 4) FTP login on the donor host using the regular account.
- 5) Copy the file from the donor host to the RELAY host.
- 6) End the FTP session, and disconnect from the donor host.
- 7) Logout of the RELAY host, close the Telnet/TCP connection, and quit Telnet on the local host.



Phase 2:

- 1) Use the local user FTP to connect via FTP/TCP to the RELAY host.
- 2) FTP login on the RELAY host using the special account for the relay service.
- 3) Copy the file from the RELAY host to the local host, and delete the file from the RELAY host.
- 4) End the FTP session, and disconnect from the RELAY host.



Note that the relay host may have a policy of deleting files more than a few hours or days old.

APPENDIX C. Mail Relay Scenario

Suppose a user on a TCP-only host wishes to send a message to a user on an NCP-only host which has implemented SMTP.

- 1) Use the local mail composition program to prepare the message. Address the message to the recipient at his or her host. Tell the composition program to queue the message.
- 2) The background mailer-daemon finds the queued message. It checks the destination host name in a table to find the internet address. Instead it finds that the destination host is a NCP-only host. The mailer-daemon then checks a list of mail RELAY hosts and selects one. It send the message to the selected mail RELAY host using the SMTP procedure.
- 3) The mail RELAY host accepts the message for relaying. It checks the destination host name and discovers that it is a NCP-only host which has implemented SMTP. The mail RELAY host then sends the message to the destination using the SMTP/NCP procedure.



Suppose a user on a TCP-only host wishes to send a message to a user on an NCP-only non-SMTP host.

- 1) Use the local mail composition program to prepare the message. Address the message to the recipient at his or her host. Tell the composition program to queue the message.
- 2) The background mailer-daemon finds the queued message. It checks the destination host name in a table to find the internet address. Instead it finds that the destination host is a NCP-only host. The mailer-daemon then checks a list of mail RELAY hosts and selects one. It send the message to the selected mail RELAY host using the SMTP procedure.
- 3) The mail RELAY host accepts the message for relaying. It checks the destination host name and discovers that it is a NCP-only non-SMTP host. The mail RELAY host then sends the message to the destination using the old FTP/NCP mail procedure.



Suppose a user on a NCP-only non-SMTP host wishes to send a message to a user on an TCP-only host. Suppose the destination user is "Smith" and the host is "ABC-X".

- 1) Use the local mail composition program to prepare the message. Address the message to "Smith.ABC-X@FORWARDER". Tell the composition program to queue the message.
- 2) The background mailer-daemon finds my queued message. It sends the message to host FORWARDER using the old FTP/NCP mail procedure.
- 3) The special forwarder host converts the "user name" supplied by the FTP/NCP mail procedure (in the MAIL or MLFL command) to "Smith@ABC-X" (in the SMTP RCTP command) and queues the message to be processed by the SMTP mailer-daemon program on this same host. No conversion of the mailbox addresses in made in the message header or body.
- 4) The SMTP mailer-daemon program on the forwarder host finds this queued message and checks the destination host name in a table to find the internet address. It finds the destination address and send the mail using the SMTP procedure.



APPENDIX D. IP/TCP Implementation Status

Please note that the information in this section may become quickly dated. Current information on the status of IP and TCP implementations can be obtained from the file <INTERNET-NOTEBOOK>TCP-IP-STATUS.TXT on ISIF.

BBN C/70 UNIX

Date: 18 Nov 1981
From: Rob Gurwitz <gurwitz at BBN-RSM>

The C/70 processor is a BBN-designed system with a native instruction set oriented toward executing the C language. It supports UNIX Version 7 and provides for user processes with a 20-bit address space. The TCP/IP implementation for the C/70 was ported from the BBN VAX TCP/IP, and shares all of its features.

This version of TCP/IP is running experimentally at BBN, but is still under development. Performance tuning is underway, to make it more compatible with the C/70's memory management system.

BBN GATEWAYS

Date: 19 Nov 1981
From: Alan Sheltzer <sheltzer at BBN-UNIX>

In an effort to provide improved service in the gateways controlled by BBN, a new gateway implementation written in macro-11 instead of BCPL is being developed. The macro-11 gateway will provide users with internet service that is functionally equivalent to that provided by the current BCPL gateways with some performance improvements.

ARPANET/SATNET gateway at BBN (10.3.0.40),
ARPANET/SATNET gateway at NDRE (10.3.0.41),
Comsat DCN Net/SATNET gateway at COMSAT (4.0.0.39),
SATNET/UCL Net/RSRE Net gateway at UCL (4.0.0.60),
PR Net/RCC Net gateway at BBN (3.0.0.62),
PR Net/ARPANET gateways at SRI (10.3.0.51, 10.1.0.51),
PR Net/ARPANET gateway at Ft. Bragg (10.0.0.38).

BBN H316 and C/30 TAC

Date: 18 November 1981
From: Bob Hinden <Hinden@BBN-UNIX>

The Terminal Access Controller (TAC) is user Telnet host that supports TCP/IP and NCP host to host protocols. It runs in 32K H-316 and 64K C/30 computers. It supports up to 63 terminal ports. It connects to a network via an 1822 host interface.

For more information on the TAC's design, see IEN-166.

BBN HP-3000

Date: 14 May 1981
From: Jack Sax <sax@BBN-UNIX>

The HP3000 TCP code is in its final testing stages. The code includes under the MPE IV operating system as a special high priority process. It is not a part of the operating system kernel because MPE IV has no kernel. The protocol process includes TCP, IP, 1822 and a new protocol called HDH which allows 1822 messages to be sent over HDLC links. The protocol process has about 8k bytes of code and at least 20k bytes of data depending on the number of buffers allocated.

In addition to the TCP the HP3000 has user and server TELNET as well as user FTP. A server FTP may be added later.

A complete description of the implementation software can be found in IEN-167.

BBN PDP-11 UNIX

Date: 14 May 1981
From: Jack Haverty <haverty@BBN-UNIX>

This TCP implementation was written in C. It runs as a user process in version 6 UNIX, with modifications added by BBN for network access. It supports user and server Telnet.

This implementation was done under contract to DCEC. It is installed currently on several PDP-11/70s and PDP-11/44s. Contact Ed Cain at DCEC <cain@EDN-UNIX> for details of further development.

BBN TENEX & TOPS20

Date: 23 Nov 1981
From: Charles Lynn <CLynn@BBNA>

TCP4 and IP4 are available for use with the TENEX operating system running on a Digital KA10 processor with BBN pager. TCP4 and IP4 are also available as part of TOPS20 Release 3A and Release 4 for the Digital KL10 and KL20 processors.

Above the IP layer, there are two Internet protocols within the monitor itself (TCP4 and GGP). In addition up to eight (actually a monitor assembly parameter) protocols may be implemented by user-mode programs via the "Internet User Queue" interface. The GGP or Gateway-Gateway Protocol is used to receive advice from Internet Gateways in order to control message flow. The GGP code is in the process of being changed and the ICMP protocol is being added.

TCP4 is the other monitor-supplied protocol and it has two types of connections -- normal data connections and "TCP Virtual Terminal" (TVT) connections. The former are used for bulk data transfers while the latter provide terminal access for remote terminals.

Note that TVTs use the standard ("New") TELNET protocol. This is identical to that used on the ARPANET with NCP and in fact, is largely implemented by the same code.

Performance improvements, support for the new address formats, and User and Server FTP processes above the TCP layer are under development.

BBN VAX UNIX

Date: 18 Nov 1981
From: Rob Gurwitz <gurwitz at BBN-RSM>

The VAX TCP/IP implementation is written in C for Berkeley 4.1BSD UNIX, and runs in the UNIX kernel. It has been run on VAX 11/780s and 750s at several sites, and is due to be generally available in early 1982.

The implementation conforms to the TCP and IP specifications (RFC 791, 793). The implementation supports the new extended internet address formats, and both GGP and ICMP. It also supports multiple network access protocols and device drivers. Aside from ARPANET 1822 and the AOC LH/DH-11 driver, experimental drivers have also been developed for ETHERNET. There are user interfaces for

accessing the IP and local network access layers independent of the TCP.

Higher level protocol services include user and server TELNET, MTP, and FTP, implemented as user level programs. There are also tools available for monitoring and recording network traffic for debugging purposes.

Continuing development includes performance enhancements. The implementation is described in IEN-168.

COMSAT

Date: 30 Apr 1980
From: Dave Mills <Mills@ISIE>

The TCP/IP implementation here runs in an LSI-11 with a homegrown operating system compatible in most respects to RT-11. Besides the TCP/IP levels the system includes many of the common high-level protocols used in the ARPANET community, such as TELNET, FTP and XNET.

UDEC PDP-11 UNIX

Date: 23 Nov 1981
From: Ed Cain <cain@EDN-UNIX>

This TCP/IP/ICMP implementation runs as a user process in version 6 UNIX, with modifications obtained from BBN for network access. IP reassembles fragments into datagrams, but has no separate IP user interface. TCP supports user and server Telnet, echo, discard, internet mail, and a file transfer service. ICMP generates replies to Echo Requests, and sends Source-Quench when reassembly buffers are full.

Hardware - PDP-11/70 and PDP-11/45 running UNIX version 6, with BBN IPC additions. Software - written in C, requiring 25K instruction space, 20K data space. Supports 10 connections.

DTI VAX

Date: 15 May 1981
From: Gary Grossman <grg@DTI>

Digital Technology Incorporated (DTI) IP/TCP for VAX/VMS

The following describes the IP and TCP implementation that DTI plans to begin marketing in 4th Quarter 1981 as part of its VAX/VMS network software package.

Hardware: VAX-11/780 or /750. Operating System: DEC standard VAX/VMS Release 2.0 and above. Implementation Language: Mostly C, with some MACRO. Connections supported: Maximum of 64.

User level protocols available: TELNET, FTP, and MTP will be available. (The NFE version uses AUTODIN II protocols.)

MIT MULTICS

Date: 13 May 1981
From: Dave Clark <Clark@MIT-Multics>

Multics TCP/IP is implemented in PL/1 for the HISI 68/80. It has been in experimental operation for about 18 months; it can be distributed informally as soon as certain modifications to the system are released by Honeywell. The TCP and IP package are currently being tuned for performance, especially high throughput data transfer.

Higher level services include user and server telnet, and a full function MTP mail forwarding package.

The TCP and IP contain good logging and debugging facilities, which have proved useful in the checkout of other implementations. Please contact us for further information.

SRI LSI-11

Date: 15 May 1981
From: Jim Mathis <mathis.tscb@Sri-Unix>

The IP/TCP implementation for the Packet Radio terminal interface unit is intended to run on an LSI-11 under the MOS real-time operating system. The TCP is written in MACRO-11 assembler language. The IP is currently written in assembler language; but is being converted into C. There are no plans to convert the TCP from assembler into C.

The TCP implements the full specification. The TCP appears to be functionally compatible with all other major implementations. In particular, it is used on a daily basis to provide communications between users on the Ft. Bragg PRNET and ISID on the ARPANET.

The IP implementation is reasonably complete, providing fragmentation and reassembly; routing to the first gateway; and a complete host-side GGP process.

A measurement collection mechanism is currently under development to collect TCP and IP statistics and deliver them to a measurement host for data reduction.

UCLA IBM

Date: 13 May 1981
From: Bob Braden <Braden@ISIA>

Hardware: IBM 360 or 370, with a "Santa Barbara" interface to the IMP.

Operating System: OS/MVS with ACF/VTAM. An OS/MVT version is also available. The UCLA NCP operates as a user job, with its own internal multiprogramming and resource management mechanisms.

Implementation Language: BAL (IBM's macro assembly language)

User-Level Protocols Available: User and Server Telnet

IEN-48

**THE CATENET MODEL
FOR INTERNETWORKING**

July 1978

(25)

THE CATENET MODEL FOR INTERNETWORKING

Introduction

The term "catenet" was introduced by L. Pouzin in 1974 in his early paper on packet network interconnection [1]. The U.S. DARPA research project on this subject has adopted the term to mean roughly "the collection of packet networks which are connected together." This is, however, not a sufficiently explicit definition to determine, for instance, whether a new network is in conformance with the rules for network interconnection which make the catenet function as confederation of co-operating networks. This paper attempts to define the objectives and limitations of the ARPA-internetworking project and to make explicit the catenet model on which the internetworking strategy is based.

Objectives

The basic objective of this project is to establish a model and a set of rules which will allow data networks of widely varying internal operation to be interconnected, permitting users to access remote resources and to permit intercomputer communication across the connected networks.

One motivation for this objective is to permit the internal technology of a data network to be optimized for local operation but also permit these locally optimized nets to be readily interconnected into an organized catenet. The term "local" is used in a loose sense, here, since it means "peculiar to the particular network" rather than "a network of limited geographic extent." A satellite-based network such as the ARPA packet satellite network therefore has "local" characteristics (e.g., broadcast operation) even though it spans many thousands of square miles geographically speaking.

A second motivation is to allow new networking technology to be introduced into the existing catenet while remaining functionally compatible with existing systems. This allows for the phased introduction of new and obsolescence of old networks without requiring a global simultaneous change.

Assumptions

One of the first questions which must be settled in a project of this sort is "what types of data networks should be included in the catenet model?" The answer to this question is rooted in the basic functionality of each candidate network. Each network is assumed to support the attachment of a collection of programmable computers. Our essential assumption is that any participating data network can carry a datagram containing no less than 1000

bits of data not including a local network header containing local control information. Furthermore, it is assumed that the participating network allows switched access so that any source computer can quickly enter datagrams for successive and different destination computers with little or no delay (i.e., on the order of tens of milliseconds or less switching time).

Under these assumptions, we can readily include networks which offer "datagram" interfaces to subscribing host computers. That is, the switching is done by the network based on a destination address contained in each datagram passing across the host to network interface.

The assumptions do not rule out virtual circuit interface networks, nor do they rule out very fast digital circuit switching networks. In these cases, the important functionality is still that a datagram can be carried over a real or virtual circuit from source to destination computer, and that the switching delay is below a few tens of milliseconds.

An important administrative assumption is that the format of an internet datagram can be commonly agreed, along with a common internet addressing plan. The basic assumption regarding datagram transport within any particular network is that the datagram will be carried, embedded in one or more packets, or frames, across the network. If fragmentation and reassembly of datagrams occurs within a network it is invisible for purposes of the catenet model. Provision is also made in the datagram format for the fragmentation of datagrams into smaller, but identically structured datagrams which can be carried independently across any particular network. No a priori position is taken regarding the choice between internal (invisible) fragmentation and reassembly or external (visible) fragmentation. This is left to each network to decide. We will return to the topic of datagram format and addressing later.

It is very important to note that it is explicitly assumed that datagrams are not necessarily kept in the same sequence on exiting a network as when they entered. Furthermore, it is assumed that datagrams may be lost or even duplicated within the network. It is left up to higher level protocols in the catenet model to recover from any problems these assumptions may introduce. These assumptions do not rule out data networks which happen to keep datagrams in sequence.

It is also assumed that networks are interconnected to each other by means of a logical "gateway." As the definition of the gateway concept unfolds, we will see that certain types of network interconnections are "invisible" with respect to the catenet model. All gateways which are visible to the catenet model have the characteristic that they can interpret the address

fields of internet datagrams so as to route them to other gateways or to destinations within the networks directly attached to (or associated with) the gateway. To send a datagram to a destination, a gateway may have to map an internet address into a local network address and embed the datagram in one or more local network packets before injecting it into the local network for transport.

The set of catenet gateways are assumed to exchange with each other at least a certain minimum amount of information to enable routing decisions to be made, to isolate failures and identify errors, and to exercise internet flow and congestion control. Furthermore, it is assumed that each catenet gateway can report a certain minimum amount of status information to an internetwork monitoring center for the purpose of identifying and isolating catenet failures, collecting minimal performance statistics and so on.

A subset of catenet gateways may provide access control enforcement services. It is assumed that a common access control enforcement mechanism is present in any catenet gateway which provides this service. This does not rule out local access control imposed by a particular network. But to provide globally consistent access control, commonality of mechanism is essential.

Access control is defined, at the catenet gateway, to mean "permitting traffic to enter or leave a particular network." The criteria by which entrance and exit permission are decided are the responsibility of network "access controllers" which establish access control policy. It is assumed that catenet gateways simply enforce the policy of the access controllers.

The Catenet Model

It is now possible to offer a basic catenet model of operation. Figure 1 illustrates the main components of the model. Hosts are computers which are attached to data networks. The host/network interfaces are assumed to be unique to each network. Thus, no assumptions about common network interfaces are made. A host may be connected to more than one network and it may have more than one connection to the same network, for reliability.

Gateways are shown as if they were composed of two or more "halves." Each half-gateway has two interfaces:

1. A interface to a local network.
2. An interface to another gateway-half.

One example is given of a gateway with three "halves" connecting networks A, B, and C. For modelling purposes, it is appropriate to treat this case as three pairs of gateway halves, each pair bilaterally joining a pair of networks.

The model does not rule out the implementation of monolithic gateways joining two or more nets, but all gateway functions and interactions are defined as if the gateways consisted of halves, each of which is associated with a specific network.

A very important aspect of this model is that no a priori distinction is made between a host/network interface and a gateway/network interface. Such distinctions are not ruled out, but they are not relevant to the basic catenet model.

As a consequence, the difference between a host which is connected to two networks and a monolithic gateway between networks is entirely a matter of whether table entries in other gateways identify the host as a gateway, and whether the standard gateway functionality exists in the host. If no other gateway or host recognizes the dual net host as a gateway or if the host cannot pass datagrams transparently from one net to the next, then it is not considered a catenet gateway.

The model does not rule out the possibility of implementing a gateway-half entirely as part of a network switching node (e.g., as software in an ARPANET IMP). The important aspect of gateway-halves is the procedure and protocol by which the half-gateways exchange datagrams and control information.

The physical interface between directly connected gateway halves is of no special importance. For monolithic gateways, it is typically shared memory or an interprocess communication mechanism of some kind; for distinct gateway halves, it might be HDLC, VDH, any other line control procedure, or inter-computer buss mechanism.

Hidden Gateways

No explicit network hierarchy is assumed in this model. Every network is known to all catenet gateways and each catenet gateway knows how to route internet datagrams so they will eventually reach a gateway connected to the destination network.

The absence of an explicit hierarchical structure means that some network substructures may be hidden from the view of the catenet gateways. If a network is composed of a hierarchy of internal networks connected together with gateways, these "hidden gateways" will not be visible to the catenet gateways unless the internal networks are assigned global network addresses and their interconnecting gateways co-operate in the global routing and network flow control procedures.

Figure 2 illustrates a simple network hierarchy. For purposes of identification, the three catenet gateways have been labelled G(AX), G(BX) and G(CX) to indicate that these gateways join networks A and X, B and X and C and X, respectively. Only G(AX), G(BX) and G(CX) are considered catenet gateways. Thus they each are aware of networks A, B, C and X and they each exchange routing and flow-control information in a uniform way between directly connected halves.

Network X is composed of three internal networks labelled u, v and w. To distinguish them from the catenet gateways, the "hidden gateways" of net X are labelled HG(nm) where "nm" indicate which nets the hidden gateways join. For example, HG(vw) joins nets v and w. The notation for HG is symmetric, i.e., $HG(vw) = HG(wv)$.

Gateways G(AX), G(BX), G(CX) exchange connectivity and other flow control information among themselves, via network X. To do this, each gateway half must know an address, local to network X, which will allow network X to route datagrams from G(AX) to G(BX), for example.

From the figure, it is plain that G(BX) is really a host on network B and network v. But network v is not one of the globally recognized networks. Furthermore, traffic from G(AX) to G(BX) may travel from net u to net v or via nets u and w to net v. To maintain the fiction of a uniform network X, the gateway halves of G(AX), G(BX) and G(CX) attached to net X must be aware of the appropriate address strings to use to cause traffic to be routed to each catenet gateway on net X. In the next section, we outline a basic internet addressing philosophy which permits such configurations to work.

Local Gateways

Another element of the catenet model is a "local gateway" associated with each host. The local gateway is capable of reassembling fragmented internet datagrams, if necessary, and is responsible for encapsulation of internet datagrams in local network packets. The local gateway also selects internet gateways through which to route internet traffic, and responds to

routing and flow control advice from the local network and attached catenet gateways.

For example, a local gateway might encapsulate and send an internet datagram to a particular gateway on its way to a distant network. The catenet gateway might forward the packet to another gateway and send an advisory message to the local gateway recommending a change in its catenet gateway routing table. Local gateways do not participate in the general routing algorithm executed among the catenet gateways.

Internet Addressing

The basic internet datagram format is shown in Figure 3. By assumption, every network in the catenet which is recognized by the catenet gateways has a unique network number. Every host in each network is identified by a 24 bit address which is prefixed by the network number. The same host may have several addresses depending on how many nets it is connected to or how many network access lines connect it to a particular network.

For the present, it is assumed that internet addresses have the form: Net.Host. "Net" is an 8 bit network number. "Host" is a 24 bit string identifying a host on the "Net," which can be understood by catenet and possibly hidden gateways.

The catenet gateways maintain tables which allow internet addresses to be mapped into local net addresses. Local gateways do likewise, at least to the extent of mapping an "out-of-network" address into the local net address of a catenet gateway.

In general, catenet gateways maintain a table entry for each "Net" which indicates to which gateway(s) datagrams destined for that net should be sent. For each "Net" to which the gateway is attached, the gateway maintains tables, if necessary, to permit mapping from internet host addresses to local net host addresses. The typical case is that a gateway half is connected to only one network and therefore only needs to maintain local address information for a single network.

It is assumed that each network has its own locally specific addressing conventions. To simplify the translation from internet address to local address, it is advantageous, if possible, to simply concatenate a network identifier with the local "host" addresses to create an internet address. This strategy makes it potentially trivial to translate from internet to local net addresses.

More elaborate translations are possible. For example, in the case of a network with a "hidden" infrastructure, the "host" portion of the internet address could include additional structure which is understood only by catenet or hidden gateways attached to that net.

In order to limit the overhead of address fields in the header, it was decided to restrict the maximum length of the host portion of the internet address to 24 bits. The possibility of true, variable-length addressing was seriously considered. At one point, it appeared that addresses might be as long as 120 bits each for source and destination. The overhead in the higher level protocols for maintaining tables capable of dealing with the maximum possible address sizes was considered excessive.

For all the networks presently expected to be a part of the experiment, 24 bit host addresses are sufficient, even in cases where a transformation other than the trivial concatenation of local host address with network address is needed to form the 32 bit internet host address.

One of the major arguments in favor of variable length "addressing" is to support what is called "source-routing." The structure of the information in the "address" really identifies a route (e.g., through a particular sequence of networks and gateways). Such a capability could support ad hoc network interconnections in which a host on two nets could serve as a private gateway. Though it would not participate in catenet routing or flow control procedures, any host which knows of this private gateway could send "source-routed" internet datagrams to that host.

To support experiments with source routing, the internet datagram includes a special option which allows a source to specify a route. The option format is illustrated in Figure 4. The option code identifies the option and the length determines its extent. The pointer field indicates which intermediate destination address should be reached next in the source-selected route.

Source routing can be used to allow ad hoc network interconnections to occur before a new net has been assigned a global network identifier.

In general, catenet gateways can only interpret internet addresses of the form Net.Host. Private gateways could interpret other, local addresses for desired destinations. If a source knew the local addresses of each intermediate private gateway, it could construct a source-route which is the concatenation of the local addresses of each intermediate host.

Local and internet addresses could be inter-mixed in a single source route as long as catenet gateways only had to interpret full internet addresses when the source-routed datagram appeared for servicing. Private gateways could interpret local and internet addresses, as desired.

Since the source or destination of a source-routed datagram may not have an internet address, it may be necessary to provide a return route for replies. This might be done by modifying the content of the original route to contain "back Pointer" to intermediate destinations. Note that the local address of a private gateway in one network is usually different from its local address in the adjacent network.

Typically, a source would create a route which contains first the internet address of the host or gateway nearest to the desired destination. The next address in the route would be the local address of the destination. Figure 5 illustrates this notion. Host A.a wants to communicate with host Z. But Z is not attached to a formally recognized network.

To achieve its goal, host A.a can emit source-routed packets with the route: "B.y, Z." B.y identifies the host (private gateway) between net B and the new network as the first intermediate stop. The private gateway uses the "Z" information to deliver the datagram to the destination. When the datagram arrives, its route should contain "y.A.a" if the private gateway knows how to interpret A.a or "y, W, A.a" if the private gateway only knows about addresses local to network B.

Other Issues

The catenet model should provide for error messages originating within a network to be carried usefully back to the source. A global encoding of error messages or status messages is needed.

It is assumed that the gateway halves of a given network have a common status reporting, flow and congestion control mechanism. However, the halves on different nets may operate differently. There should be a defined interface between gateway halves which permits internet flow, congestion and error control to be exercised.

A gateway monitoring center [3] is postulated which can collect, correlate and display current gateway status. Such a center should not be required for the internet protocols to function, but could be used to manage an internet environment.

Accounting, accountability and access control procedures should be defined for the global catenet.

References

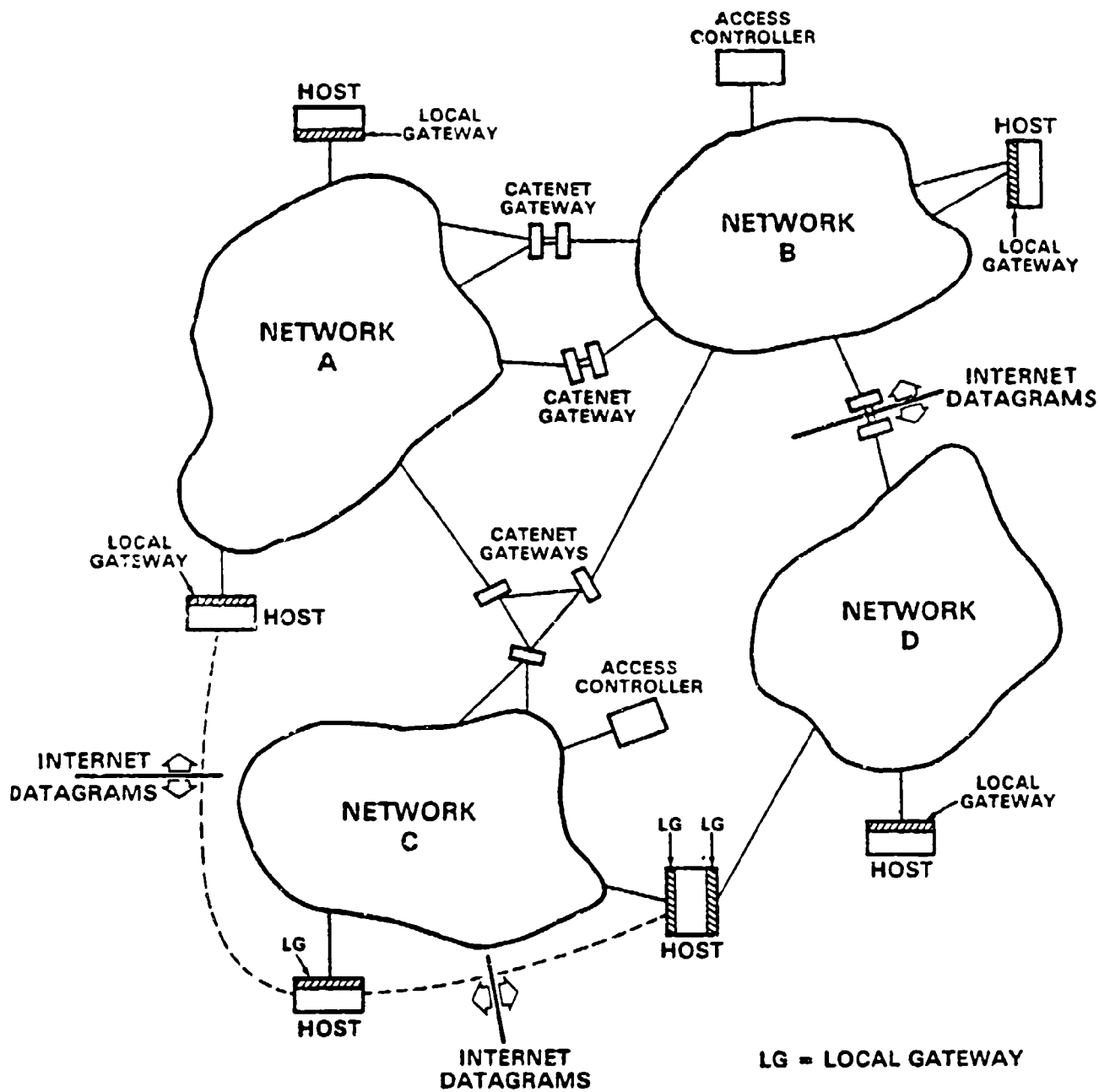
1. Pouzin, L., "A Proposal for Interconnecting Packet Switching Networks," Proceedings of EUROCOMP, Brunel University, May 1974, pp. 1023-36.
2. Postel, J. "Internetwork Datagram Protocol Specification," Version 4, Internetwork Experiment Note No. 41, Section 2.3.2.1, Internet Experiment Notebook, June 1978.
3. Davidson, John, "CATENET MONITORING AND CONTROL: A model for the Gateway Component," IEN #32, Section 2.3.3.12, Internet Notebook, April 1978.

NOTE: The figures are not included in the online version. They may be obtained from:

Jon Postel
USC - Information Sciences Institute
Suite 1100
4676 Admiralty Way
Marina del Rey, California 90291

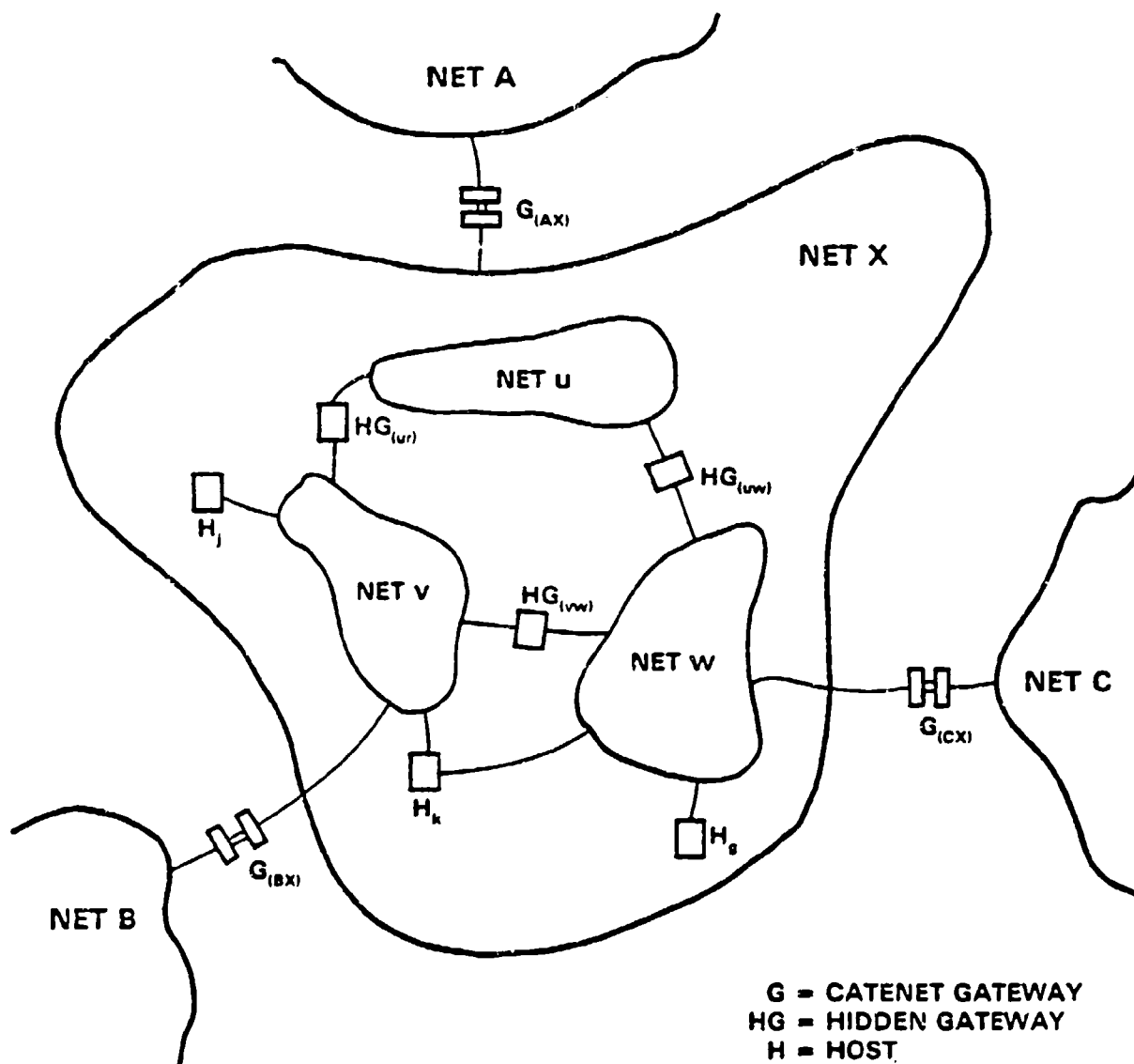
Phone: (213) 822-1511

ARPANET Mailbox: POSTEL@ISIF



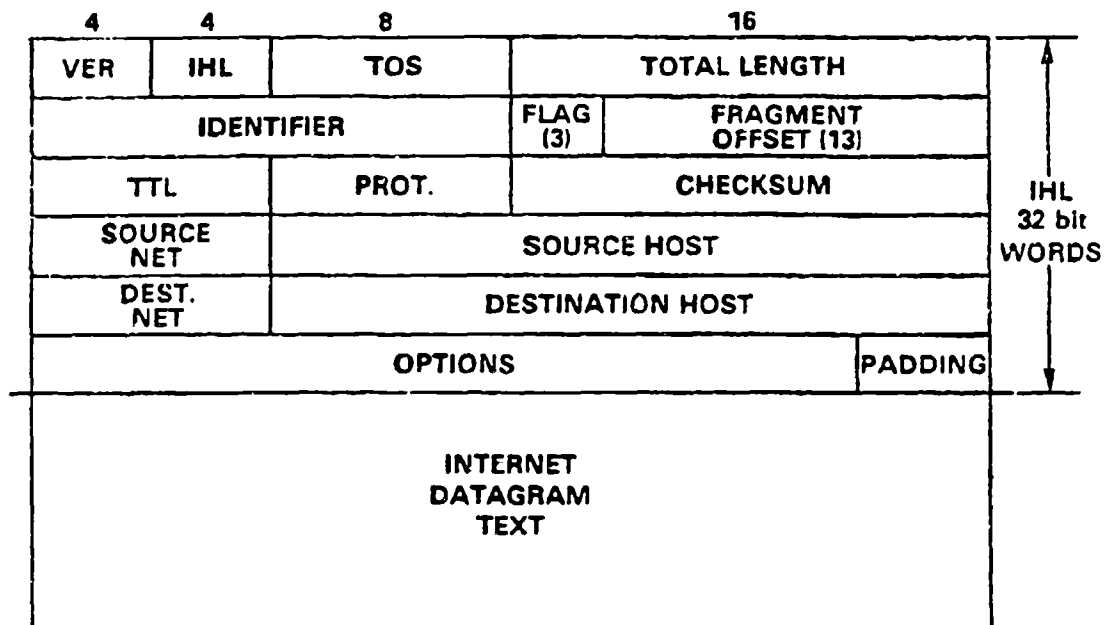
BASIC CATENET MODEL

Figure 1



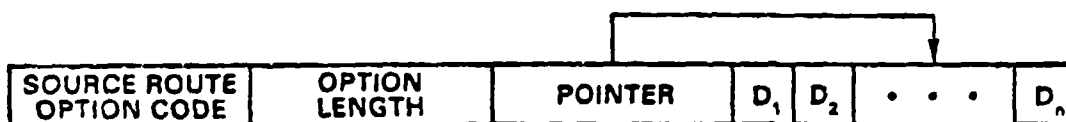
HIDDEN GATEWAYS IN HIERARCHICAL NETWORKS

Figure 2

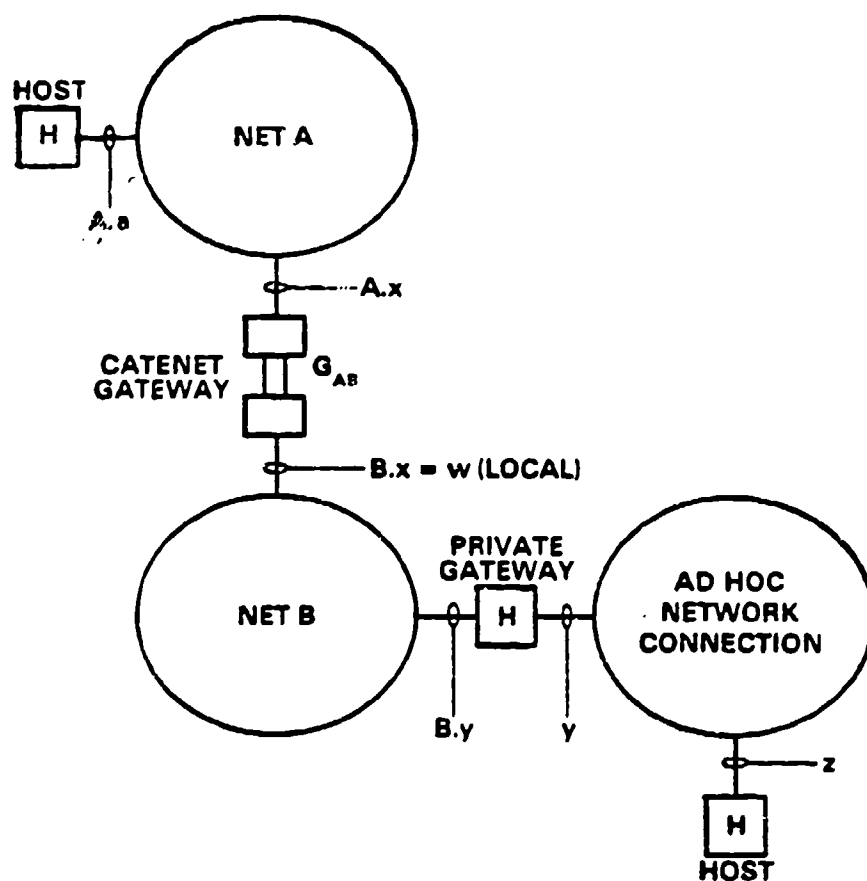


- VER = VERSION TYPE
- IHL = INTERNET HEADER LENGTH IN 32 bit WORDS
- TOS = TYPE OF INTERNET SERVICE DESIRED
e.g. "LOW DELAY", "LOW COST", "HIGH BANDWIDTH",
"HIGH RELIABILITY", "DON'T DISCARD".
- FLAG = CONTROL INDICATIONS SUCH AS
"OPTIONS PRESENT", "MORE FRAGMENTS".
- PROT = PROTOCOL IDENTIFIER (e.g. TCP, REAL-TIME)

INTERNET DATAGRAM FORMAT
FIGURE 3



SOURCE ROUTE OPTION FORMAT
FIGURE 4



**PRIVATE GATEWAYS AND SOURCE ROUTING
FIGURE 5**

NETWORK LEVEL

INTERNET PROTOCOL

**INTERNET CONTROL
MESSAGE PROTOCOL**

RFC-791

INTERNET PROTOCOL

September 1981

RFC: 791

INTERNET PROTOCOL

**DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION**

September 1981

prepared for

**Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209**

by

**Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291**

TABLE OF CONTENTS

PREFACE	ii
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Scope	1
1.3 Interfaces	1
1.4 Operation	2
2. OVERVIEW	5
2.1 Relation to Other Protocols	9
2.2 Model of Operation	5
2.3 Function Description	7
2.4 Gateways	9
3. SPECIFICATION	11
3.1 Internet Header Format	11
3.2 Discussion	23
3.3 Interfaces	31
APPENDIX A: Examples & Scenarios	34
APPENDIX B: Data Transmission Order	39
GLOSSARY	41
REFERENCES	45

September 1981

Internet Protocol

PREFACE

This document specifies a revised version of the DoD Standard Internet Protocol. This document is based on six earlier editions of the ARPA Internet Protocol Specification, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition revises aspects of addressing, error handling, option codes, and the security, precedence, compartments, and handling restriction features of the internet protocol.

Jon Postel

Editor

[Page 11]

(53)

September 1981

RFC: 791
Replaces: RFC 760
IENs 128, 123, 111,
80, 54, 44, 41, 28, 26

INTERNET PROTOCOL

DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION

1. INTRODUCTION

1.1. Motivation

The Internet Protocol is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet" [1]. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

1.2. Scope

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

1.3. Interfaces

This protocol is called on by host-to-host protocols in an internet environment. This protocol calls on local network protocols to carry the internet datagram to the next gateway or destination host.

For example, a TCP module would call on the internet module to take a TCP segment (including the TCP header and user data) as the data portion of an internet datagram. The TCP module would provide the addresses and other parameters in the internet header to the internet module as arguments of the call. The internet module would then create an internet datagram and call on the local network interface to transmit the internet datagram.

In the ARPANET case, for example, the internet module would call on a

[Page 1]

(55)

September 1981

Internet Protocol Introduction

local net module which would add the 1822 leader [2] to the internet datagram creating an ARPANET message to transmit to the IMP. The ARPANET address would be derived from the internet address by the local network interface and would be the address of some host in the ARPANET, that host might be a gateway to other networks.

1.4. Operation

The internet protocol implements two basic functions: addressing and fragmentation.

The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing.

The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The model of operation is that an internet module resides in each host engaged in internet communication and in each gateway that interconnects networks. These modules share common rules for interpreting address fields and for fragmenting and assembling internet datagrams. In addition, these modules (especially in gateways) have procedures for making routing decisions and other functions.

The internet protocol treats each internet datagram as an independent entity unrelated to any other internet datagram. There are no connections or logical circuits (virtual or otherwise).

The internet protocol uses four key mechanisms in providing its service: Type of Service, Time to Live, Options, and Header Checksum.

The Type of Service is used to indicate the quality of the service desired. The type of service is an abstract or generalized set of parameters which characterize the service choices provided in the networks that make up the internet. This type of service indication is to be used by gateways to select the actual transmission parameters for a particular network, the network to be used for the next hop, or the next gateway when routing an internet datagram.

The Time to Live is an indication of an upper bound on the lifetime of an internet datagram. It is set by the sender of the datagram and reduced at the points along the route where it is processed. If the time to live reaches zero before the internet datagram reaches its destination, the internet datagram is destroyed. The time to live can be thought of as a self destruct time limit.

September 1981

Internet Protocol
Introduction

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The options include provisions for timestamps, security, and special routing.

The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error.

The internet protocol does not provide a reliable communication facility. There are no Acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control.

Errors detected may be reported via the Internet Control Message Protocol (ICMP) [3] which is implemented in the internet protocol module.

Internet Protocol

September 1981

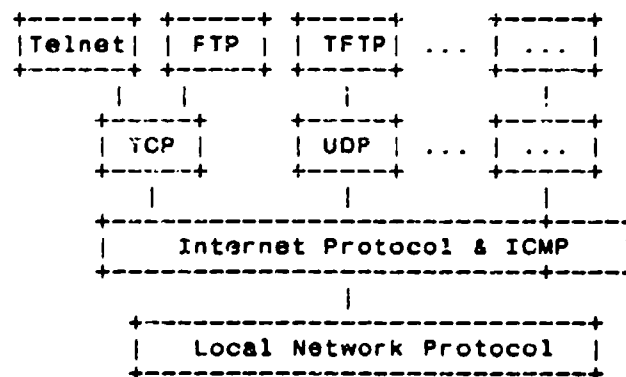
REFERENCES

- [1] Cerf, V., "The Catenet Model for Internetworking," Information Processing Techniques Office, Defense Advanced Research Projects Agency, IEN 48, July 1978.
- [2] Bolt Beranek and Newman, "Specification for the Interconnection of a Host and an IMP," BBN Technical Report 1822, Revised May 1978.
- [3] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification," RFC 792, USC/Information Sciences Institute, September 1981.
- [4] Shoch, J., "Inter-Network Naming, Addressing, and Routing," COMPCON, IEEE Computer Society, Fall 1978.
- [5] Postel, J., "Address Mappings," RFC 796, USC/Information Sciences Institute, September 1981.
- [6] Shoch, J., "Packet Fragmentation in Inter-Network Protocols," Computer Networks, v. 3, n. 1, February 1979.
- [7] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.
- [8] Postel, J., "Service Mappings," RFC 795, USC/Information Sciences Institute, September 1981.
- [9] Postel, J., "Assigned Numbers," RFC 790, USC/Information Sciences Institute, September 1981.

2. OVERVIEW

2.1. Relation to Other Protocols

The following diagram illustrates the place of the internet protocol in the protocol hierarchy:



Protocol Relationships

Figure 1.

Internet protocol interfaces on one side to the higher level host-to-host protocols and on the other side to the local network protocol. In this context a "local network" may be a small network in a building or a large network such as the ARPANET.

2.2. Model of Operation

The model of operation for transmitting a datagram from one application program to another is illustrated by the following scenario:

We suppose that this transmission will involve one intermediate gateway.

The sending application program prepares its data and calls on its local internet module to send that data as a datagram and passes the destination address and other parameters as arguments of the call.

The internet module prepares a datagram header and attaches the data to it. The internet module determines a local network address for this internet address, in this case it is the address of a gateway.

September 1981

Internet Protocol Overview

It sends this datagram and the local network address to the local network interface.

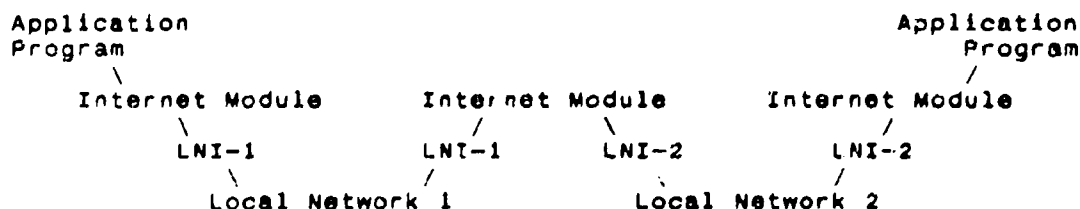
The local network interface creates a local network header, and attaches the datagram to it, then sends the result via the local network.

The datagram arrives at a gateway host wrapped in the local network header, the local network interface strips off this header, and turns the datagram over to the internet module. The internet module determines from the internet address that the datagram is to be forwarded to another host in a second network. The internet module determines a local net address for the destination host. It calls on the local network interface for that network to send the datagram.

This local network interface creates a local network header and attaches the datagram sending the result to the destination host.

At this destination host the datagram is stripped of the local net header by the local network interface and handed to the internet module.

The internet module determines that the datagram is for an application program in this host. It passes the data to the application program in response to a system call, passing the source address and other parameters as results of the call.



Transmission Path

Figure 2

2.3. Function Description

The function or purpose of Internet Protocol is to move datagrams through an interconnected set of networks. This is done by passing the datagrams from one internet module to another until the destination is reached. The internet modules reside in hosts and gateways in the internet system. The datagrams are routed from one internet module to another through individual networks based on the interpretation of an internet address. Thus, one important mechanism of the internet protocol is the internet address.

In the routing of messages from one internet module to another, datagrams may need to traverse a network whose maximum packet size is smaller than the size of the datagram. To overcome this difficulty, a fragmentation mechanism is provided in the internet protocol.

Addressing

A distinction is made between names, addresses, and routes [4]. A name indicates what we seek. An address indicates where it is. A route indicates how to get there. The internet protocol deals primarily with addresses. It is the task of higher level (i.e., host-to-host or application) protocols to make the mapping from names to addresses. The internet module maps internet addresses to local net addresses. It is the task of lower level (i.e., local net or gateways) procedures to make the mapping from local net addresses to routes.

Addresses are fixed length of four octets (32 bits). An address begins with a network number, followed by local address (called the "rest" field). There are three formats or classes of internet addresses: in class a, the high order bit is zero, the next 7 bits are the network, and the last 24 bits are the local address; in class b, the high order two bits are one-zero, the next 14 bits are the network and the last 18 bits are the local address; in class c, the high order three bits are one-one-zero, the next 21 bits are the network and the last 11 bits are the local address.

Care must be taken in mapping internet addresses to local net addresses; a single physical host must be able to act as if it were several distinct hosts to the extent of using several distinct internet addresses. Some hosts will also have several physical interfaces (multi-homing).

That is, provision must be made for a host to have several physical interfaces to the network with each having several logical internet addresses.

September 1981

Examples of address mappings may be found in "Address Mappings" [5].

Fragmentation

Fragmentation of an internet datagram is necessary when it originates in a local net that allows a large packet size and must traverse a local net that limits packets to a smaller size to reach its destination.

An internet datagram can be marked "don't fragment." Any internet datagram so marked is not to be internet fragmented under any circumstances. If internet datagram marked don't fragment cannot be delivered to its destination without fragmenting it, it is to be discarded instead.

Fragmentation, transmission and reassembly across a local network which is invisible to the internet protocol module is called intranet fragmentation and may be used [6].

The internet fragmentation and reassembly procedure needs to be able to break a datagram into an almost arbitrary number of pieces that can be later reassembled. The receiver of the fragments uses the identification field to ensure that fragments of different datagrams are not mixed. The fragment offset field tells the receiver the position of a fragment in the original datagram. The fragment offset and length determine the portion of the original datagram covered by this fragment. The more-fragments flag indicates (by being reset) the last fragment. These fields provide sufficient information to reassemble datagrams.

The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

To fragment a long internet datagram, an internet protocol module (for example, in a gateway), creates two new internet datagrams and copies the contents of the internet header fields from the long datagram into both new internet headers. The data of the long datagram is divided into two portions on a 8 octet (64 bit) boundary (the second portion might not be an integral multiple of 8 octets, but the first must be). Call the number of 8 octet blocks in the first portion NFB (for Number of Fragment Blocks). The first portion of the data is placed in the first new internet datagram, and the total length field is set to the length of the first

datagram. The more-fragments flag is set to one. The second portion of the data is placed in the second new internet datagram, and the total length field is set to the length of the second datagram. The more-fragments flag carries the same value as the long datagram. The fragment offset field of the second new internet datagram is set to the value of that field in the long datagram plus NFB.

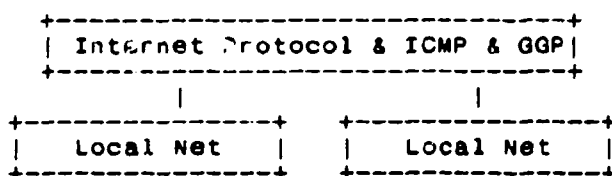
This procedure can be generalized for an n-way split, rather than the two-way split described.

To assemble the fragments of an internet datagram, an internet protocol module (for example at a destination host) combines internet datagrams that all have the same value for the four fields: identification, source, destination, and protocol. The combination is done by placing the data portion of each fragment in the relative position indicated by the fragment offset in that fragment's internet header. The first fragment will have the fragment offset zero, and the last fragment will have the more-fragments flag reset to zero.

2.4. Gateways

Gateways implement internet protocol to forward datagrams between networks. Gateways also implement the Gateway to Gateway Protocol (GGP) [7] to coordinate routing and other internet control information.

In a gateway the higher level protocols need not be implemented and the GGP functions are added to the IP module.



Gateway Protocols

Figure 3.

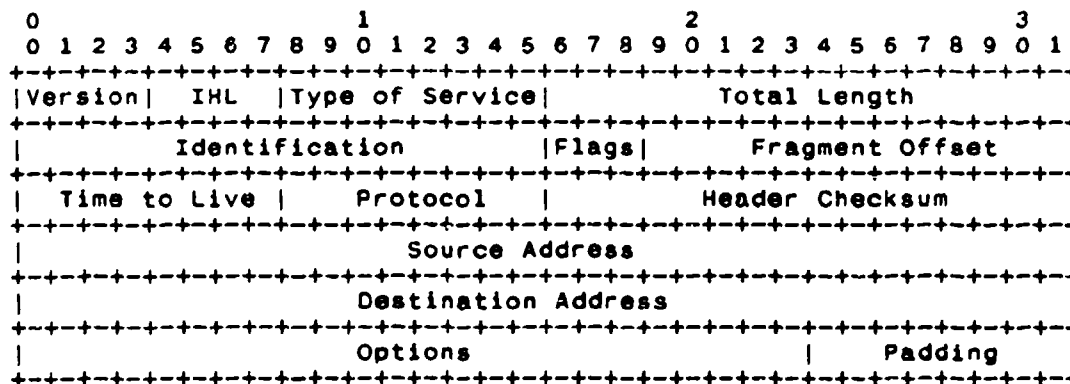
Internet Protocol

September 1981

3. SPECIFICATION

3.1. Internet Header Format

A summary of the contents of the internet header follows:



Example Internet Datagram Header

Figure 4.

Note that each tick mark represents one bit position.

Version: 4 bits

The Version field indicates the format of the internet header. This document describes version 4.

IHL: 4 bits

Internet Header Length is the length of the internet header in 32 bit words, and thus points to the beginning of the data. Note that the minimum value for a correct header is 5.

September 1981

Type of Service: 8 bits

The Type of Service provides an indication of the abstract parameters of the quality of service desired. These parameters are to be used to guide the selection of the actual service parameters when transmitting a datagram through a particular network. Several networks offer service precedence, which somehow treats high precedence traffic as more important than other traffic (generally by accepting only traffic above a certain precedence at time of high load). The major choice is a three way tradeoff between low-delay, high-reliability, and high-throughput.

Bits 0-2: Precedence.

Bit 3: 0 = Normal Delay, 1 = Low Delay.

Bits 4: 0 = Normal Throughput, 1 = High Throughput.

Bits 5: 0 = Normal Reliability, 1 = High Reliability.

Bit 6-7: Reserved for Future Use.

0	1	2	3	4	5	6	7
PRECEDENCE			D	T	R	O	O

Precedence

- 111 - Network Control
- 110 - Internetwork Control
- 101 - CRITIC/ECP
- 100 - Flash Override
- 011 - Flash
- 010 - Immediate
- 001 - Priority
- 000 - Routine

The use of the Delay, Throughput, and Reliability indications may increase the cost (in some sense) of the service. In many networks better performance for one of these parameters is coupled with worse performance on another. Except for very unusual cases at most two of these three indications should be set.

The type of service is used to specify the treatment of the datagram during its transmission through the internet system. Example mappings of the internet type of service to the actual service provided on networks such as AUTODIN II, ARPANET, SATNET, and PRNET is given in "Service Mappings" [8].

September 1981

Internet Protocol
Specification

The Network Control precedence designation is intended to be used within a network only. The actual use and control of that designation is up to each network. The Internetwork Control designation is intended for use by gateway control originators only. If the actual use of these precedence designations is of concern to a particular network, it is the responsibility of that network to control the access to, and use of, those precedence designations.

Total Length: 16 bits

Total Length is the length of the datagram, measured in octets, including internet header and data. This field allows the length of a datagram to be up to 65,535 octets. Such long datagrams are impractical for most hosts and networks. All hosts must be prepared to accept datagrams of up to 576 octets (whether they arrive whole or in fragments). It is recommended that hosts only send datagrams larger than 576 octets if they have assurance that the destination is prepared to accept the larger datagrams.

The number 576 is selected to allow a reasonable sized data block to be transmitted in addition to the required header information. For example, this size allows a data block of 512 octets plus 64 header octets to fit in a datagram. The maximal internet header is 60 octets, and a typical internet header is 20 octets, allowing a margin for headers of higher level protocols.

Identification: 16 bits

An identifying value assigned by the sender to aid in assembling the fragments of a datagram.

Flags: 3 bits

Various Control Flags.

Bit 0: reserved, must be zero
Bit 1: (DF) 0 = May Fragment, 1 = Don't Fragment.
Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments.

0	1	2
+	+	+
	D	M
	F	F
+	+	+

Fragment Offset: 13 bits

This field indicates where in the datagram this fragment belongs.

September 1981

The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

Time to Live: 8 bits

This field indicates the maximum time the datagram is allowed to remain in the internet system. If this field contains the value zero, then the datagram must be destroyed. This field is modified in internet header processing. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one even if it process the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Protocol: 8 bits

This field indicates the next level protocol used in the data portion of the internet datagram. The values for various protocols are specified in "Assigned Numbers" [9].

Header Checksum: 16 bits

A checksum on the header only. Since some header fields change (e.g., time to live), this is recomputed and verified at each point that the internet header is processed.

The checksum algorithm is:

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero.

This is a simple to compute checksum and experimental evidence indicates it is adequate, but it is provisional and may be replaced by a CRC procedure, depending on further experience.

Source Address: 32 bits

The source address. See section 3.2.

Destination Address: 32 bits

The destination address. See section 3.2.

September 1981

Internet Protocol
Specification

Options: variable

The options may appear or not in datagrams. They must be implemented by all IP modules (host and gateways). What is optional is their transmission in any particular datagram, not their implementation.

In some environments the security option may be required in all datagrams.

The option field is variable in length. There may be zero or more options. There are two cases for the format of an option:

Case 1: A single octet of option-type.

Case 2: An option-type octet, an option-length octet, and the actual option-data octets.

The option-length octet counts the option-type octet and the option-length octet as well as the option-data octets.

The option-type octet is viewed as having 3 fields:

1 bit copied flag,
2 bits option class,
5 bits option number.

The copied flag indicates that this option is copied into all fragments on fragmentation.

0 = not copied
1 = copied

The option classes are:

0 = control
1 = reserved for future use
2 = debugging and measurement
3 = reserved for future use

September 1981

The following internet options are defined:

CLASS	NUMBER	LENGTH	DESCRIPTION
0	0	-	End of Option list. This option occupies only 1 octet; it has no length octet.
0	1	-	No Operation. This option occupies only 1 octet; it has no length octet.
0	2	11	Security. Used to carry Security, Compartmentation, User Group (TCC), and Handling Restriction Codes compatible with DOD requirements.
0	3	var.	Loose Source Routing. Used to route the internet datagram based on information supplied by the source.
0	9	var.	Strict Source Routing. Used to route the internet datagram based on information supplied by the source.
0	7	var.	Record Route. Used to trace the route an internet datagram takes.
0	8	4	Stream ID. Used to carry the stream identifier.
2	4	var.	Internet Timestamp.

Specific Option Definitions

End of Option List

```
+-----+
|00000000|
+-----+
Type=0
```

This option indicates the end of the option list. This might not coincide with the end of the internet header according to the internet header length. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the internet header.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

September 1981

Internet Protocol
Specification

No Operation

```
+-----+
|00000001|
+-----+
Type=1
```

This option may be used between options, for example, to align the beginning of a subsequent option on a 32 bit boundary.

May be copied, introduced, or deleted on fragmentation, or for any other reason.

Security

This option provides a way for hosts to send security, compartmentation, handling restrictions, and TCC (closed user group) parameters. The format for this option is as follows:

```
+-----+-----+---//---+---//---+---//---+---//---+
|10000010|00001011|SSS SSS|CCC CCC|HHH HHH| TCC |
+-----+-----+---//---+---//---+---//---+---//---+
Type=130 Length=11
```

Security (S field): 16 bits

Specifies one of 16 levels of security (eight of which are reserved for future use).

00000000	00000000	-	Unclassified
11110001	00110101	-	Confidential
01111000	10011010	-	EFTO
10111100	01001101	-	MMMM
01011110	00100110	-	PROG
10101111	00010011	-	Restricted
11010111	10001000	-	Secret
01101011	11000101	-	Top Secret
00110101	11100010	-	(Reserved for future use)
10011010	11110001	-	(Reserved for future use)
01001101	01111000	-	(Reserved for future use)
00100100	10111101	-	(Reserved for future use)
00010011	01011110	-	(Reserved for future use)
10001001	10101111	-	(Reserved for future use)
11000100	11010110	-	(Reserved for future use)
11100010	01101011	-	(Reserved for future use)

September 1981

Compartments (C field): 16 bits

An all zero value is used when the information transmitted is not compartmented. Other values for the compartments field may be obtained from the Defense Intelligence Agency.

Handling Restrictions (H field): 16 bits

The values for the control and release markings are alphanumeric digraphs and are defined in the Defense Intelligence Agency Manual DIAM 65-19, "Standard Security Markings".

Transmission Control Code (TCC field): 24 bits

Provides a means to segregate traffic and define controlled communities of interest among subscribers. The TCC values are trigraphs, and are available from HQ DCA Code 530.

Must be copied on fragmentation. This option appears at most once in a datagram.

Loose Source and Record Route

```
+-----+-----+-----+-----//-----+
|10000011| length | pointer|      route data      |
+-----+-----+-----+-----//-----+
Type=131
```

The loose source and record route (LSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a loose source route because the gateway or host IP is allowed to use any route of any number of other intermediate gateways to reach the next address in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Strict Source and Record Route

```

+-----+-----+-----+-----//-----+
|10001001| length | pointer|      route data      |
+-----+-----+-----+-----//-----+
Type=137

```

The strict source and record route (SSRR) option provides a means for the source of an internet datagram to supply routing information to be used by the gateways in forwarding the datagram to the destination, and to record the route information.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next source address to be processed. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A route data is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is greater than the length, the source route is empty (and the

September 1981

recorded route full) and the routing is to be based on the destination address field.

If the address in destination address field has been reached and the pointer is not greater than the length, the next address in the source route replaces the address in the destination address field, and the recorded route address replaces the source address just used, and pointer is increased by four.

The recorded route address is the internet module's own internet address as known in the environment into which this datagram is being forwarded.

This procedure of replacing the source route with the recorded route (though it is in the reverse of the order it must be in to be used as a source route) means the option (and the IP header as a whole) remains a constant length as the datagram progresses through the internet.

This option is a strict source route because the gateway or host IP must send the datagram directly to the next address in the source route through only the directly connected network indicated in the next address to reach the next gateway or host specified in the route.

Must be copied on fragmentation. Appears at most once in a datagram.

Record Route

```
+-----+-----+-----+-----//-----+
|00000111| length | pointer|      route data      |
+-----+-----+-----+-----//-----+
Type=7
```

The record route option provides a means to record the route of an internet datagram.

The option begins with the option type code. The second octet is the option length which includes the option type code and the length octet, the pointer octet, and length-3 octets of route data. The third octet is the pointer into the route data indicating the octet which begins the next area to store a route address. The pointer is relative to this option, and the smallest legal value for the pointer is 4.

A recorded route is composed of a series of internet addresses. Each internet address is 32 bits or 4 octets. If the pointer is

greater than the length, the recorded route data area is full. The originating host must compose this option with a large enough route data area to hold all the address expected. The size of the option does not change due to adding addresses. The initial contents of the route data area must be zero.

When an internet module routes a datagram it checks to see if the record route option is present. If it is, it inserts its own internet address as known in the environment into which this datagram is being forwarded into the recorded route beginning at the octet indicated by the pointer, and increments the pointer by four.

If the route data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the address into the recorded route. If there is some room but not enough room for a full address to be inserted, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

Not copied on fragmentation, goes in first fragment only.
Appears at most once in a datagram.

Stream Identifier

```

+-----+-----+-----+
|10001000|00000010|   Stream ID   |
+-----+-----+-----+
```

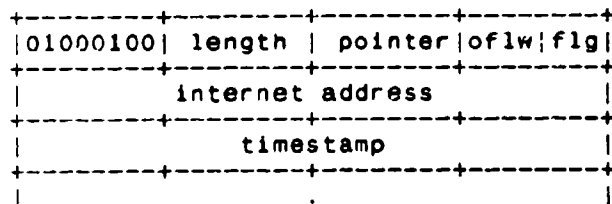
Type=136 Length=4

This option provides a way for the 16-bit SATNET stream identifier to be carried through networks that do not support the stream concept.

Must be copied on fragmentation. Appears at most once in a datagram.

September 1981

Internet Timestamp



Type = 68

The Option Length is the number of octets in the option counting the type, length, pointer, and overflow/flag octets (maximum length 40).

The Pointer is the number of octets from the beginning of this option to the end of timestamps plus one (i.e., it points to the octet beginning the space for next timestamp). The smallest legal value is 5. The timestamp area is full when the pointer is greater than the length.

The Overflow (oflw) [4 bits] is the number of IP modules that cannot register timestamps due to lack of space.

The Flag (flg) [4 bits] values are

- 0 -- time stamps only, stored in consecutive 32-bit words,
- 1 -- each timestamp is preceded with internet address of the registering entity,
- 3 -- the internet address fields are prespecified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

The Timestamp is a right-justified, 32-bit timestamp in milliseconds since midnight UT. If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time may be inserted as a timestamp provided the high order bit of the timestamp field is set to one to indicate the use of a non-standard value.

The originating host must compose this option with a large enough timestamp data area to hold all the timestamp information expected. The size of the option does not change due to adding

September 1981

Internet Protocol
Specification

timestamps. The initial contents of the timestamp data area must be zero or internet address/zero pairs.

If the timestamp data area is already full (the pointer exceeds the length) the datagram is forwarded without inserting the timestamp, but the overflow count is incremented by one.

If there is some room but not enough room for a full timestamp to be inserted, or the overflow count itself overflows, the original datagram is considered to be in error and is discarded. In either case an ICMP parameter problem message may be sent to the source host [3].

The timestamp option is not copied upon fragmentation. It is carried in the first fragment. Appears at most once in a datagram.

Padding: variable

The internet header padding is used to ensure that the internet header ends on a 32 bit boundary. The padding is zero.

3.2. Discussion

The implementation of a protocol must be robust. Each implementation must expect to interoperate with others created by different individuals. While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation must be conservative in its sending behavior, and liberal in its receiving behavior. That is, it must be careful to send well-formed datagrams, but must accept any datagram that it can interpret (e.g., not object to technical errors where the meaning is still clear).

The basic internet service is datagram oriented and provides for the fragmentation of datagrams at gateways, with reassembly taking place at the destination internet protocol module in the destination host. Of course, fragmentation and reassembly of datagrams within a network or by private agreement between the gateways of a network is also allowed since this is transparent to the internet protocols and the higher-level protocols. This transparent type of fragmentation and reassembly is termed "network-dependent" (or intranet) fragmentation and is not discussed further here.

Internet addresses distinguish sources and destinations to the host level and provide a protocol field as well. It is assumed that each protocol will provide for whatever multiplexing is necessary within a host.

Addressing

To provide for flexibility in assigning address to networks and allow for the large number of small to intermediate sized networks the interpretation of the address field is coded to specify a small number of networks with a large number of host, a moderate number of networks with a moderate number of hosts, and a large number of networks with a small number of hosts. In addition there is an escape code for extended addressing mode.

Address Formats:

High Order Bits	Format	Class
0	7 bits of net, 24 bits of host	a
10	14 bits of net, 16 bits of host	b
110	21 bits of net, 8 bits of host	c
111	escape to extended addressing mode	

A value of zero in the network field means this network. This is only used in certain ICMP messages. The extended addressing mode is undefined. Both of these features are reserved for future use.

The actual values assigned for network addresses is given in "Assigned Numbers" [9].

The local address, assigned by the local network, must allow for a single physical host to act as several distinct internet hosts. That is, there must be a mapping between internet host addresses and network/host interfaces that allows several internet addresses to correspond to one interface. It must also be allowed for a host to have several physical interfaces and to treat the datagrams from several of them as if they were all addressed to a single host.

Address mappings between internet addresses and addresses for ARPANET, SATNET, PRNET, and other networks are described in "Address Mappings" [5].

Fragmentation and Reassembly.

The internet identification field (ID) is used together with the source and destination address, and the protocol fields, to identify datagram fragments for reassembly.

The More Fragments flag bit (MF) is set if the datagram is not the last fragment. The Fragment Offset field identifies the fragment location, relative to the beginning of the original unfragmented datagram. Fragments are counted in units of 8 octets. The

September 1981

Internet Protocol
Specification

fragmentation strategy is designed so that an unfragmented datagram has all zero fragmentation information (MF = 0, fragment offset = 0). If an internet datagram is fragmented, its data portion must be broken on 8 octet boundaries.

This format allows $2^{13} = 8192$ fragments of 8 octets each for a total of 65,536 octets. Note that this is consistent with the the datagram total length field (of course, the header is counted in the total length and not in the fragments).

When fragmentation occurs, some options are copied, but others remain with the first fragment only.

Every internet module must be able to forward a datagram of 68 octets without further fragmentation. This is because an internet header may be up to 60 octets, and the minimum fragment is 8 octets.

Every internet destination must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled.

The fields which may be affected by fragmentation include:

- (1) options field
- (2) more fragments flag
- (3) fragment offset
- (4) internet header length field
- (5) total length field
- (6) header checksum

If the Don't Fragment flag (DF) bit is set, then internet fragmentation of this datagram is NOT permitted, although it may be discarded. This can be used to prohibit fragmentation in cases where the receiving host does not have sufficient resources to reassemble internet fragments.

One example of use of the Don't Fragment feature is to down line load a small host. A small host could have a boot strap program that accepts a datagram stores it in memory and then executes it.

The fragmentation and reassembly procedures are most easily described by examples. The following procedures are example implementations.

General notation in the following pseudo programs: " \leq " means "less than or equal", " \neq " means "not equal", " $=$ " means "equal", " \leftarrow " means "is set to". Also, " x to y " includes x and excludes y ; for example, "4 to 7" would include 4, 5, and 6 (but not 7).

September 1981

An Example Fragmentation Procedure

The maximum sized datagram that can be transmitted through the next network is called the maximum transmission unit (MTU).

If the total length is less than or equal the maximum transmission unit then submit this datagram to the next step in datagram processing; otherwise cut the datagram into two fragments, the first fragment being the maximum size, and the second fragment being the rest of the datagram. The first fragment is submitted to the next step in datagram processing, while the second fragment is submitted to this procedure in case it is still too large.

Notation:

FO - Fragment Offset
IHL - Internet Header Length
DF - Don't Fragment flag
MF - More Fragments flag
TL - Total Length
OFO - Old Fragment Offset
OIHL - Old Internet Header Length
OMF - Old More Fragments flag
OTL - Old Total Length
NFB - Number of Fragment Blocks
MTU - Maximum Transmission Unit

Procedure:

```
IF TL <= MTU THEN Submit this datagram to the next step
  in datagram processing ELSE IF DF = 1 THEN discard the
  datagram ELSE
  To produce the first fragment:
  (1) Copy the original internet header;
  (2) OIHL <- IHL; OTL <- TL; OFO <- FO; OMF <- MF;
  (3) NFB <- (MTU-IHL*4)/8;
  (4) Attach the first NFB*8 data octets;
  (5) Correct the header:
      MF <- 1; TL <- (IHL*4)+(NFB*8);
      Recompute Checksum;
  (6) Submit this fragment to the next step in
      datagram processing;
  To produce the second fragment:
  (7) Selectively copy the internet header (some options
      are not copied, see option definitions);
  (8) Append the remaining data;
  (9) Correct the header:
      IHL <- (((OIHL*4)-(length of options not copied))+3)/4;
```

September 1981

Internet Protocol
Specification

```
TL <- OTL - NFB*8 - (OIHL-IHL)*4);  
FO <- OFO + NFB; MF <- OMF; Recompute Checksum;  
(10) Submit this fragment to the fragmentation test; DONE.
```

In the above procedure each fragment (except the last) was made the maximum allowable size. An alternative might produce less than the maximum size datagrams. For example, one could implement a fragmentation procedure that repeatedly divided large datagrams in half until the resulting fragments were less than the maximum transmission unit size.

An Example Reassembly Procedure

For each datagram the buffer identifier is computed as the concatenation of the source, destination, protocol, and identification fields. If this is a whole datagram (that is both the fragment offset and the more fragments fields are zero), then any reassembly resources associated with this buffer identifier are released and the datagram is forwarded to the next step in datagram processing.

If no other fragment with this buffer identifier is on hand then reassembly resources are allocated. The reassembly resources consist of a data buffer, a header buffer, a fragment block bit table, a total data length field, and a timer. The data from the fragment is placed in the data buffer according to its fragment offset and length, and bits are set in the fragment block bit table corresponding to the fragment blocks received.

If this is the first fragment (that is the fragment offset is zero) this header is placed in the header buffer. If this is the last fragment (that is the more fragments field is zero) the total data length is computed. If this fragment completes the datagram (tested by checking the bits set in the fragment block table), then the datagram is sent to the next step in datagram processing; otherwise the timer is set to the maximum of the current timer value and the value of the time to live field from this fragment; and the reassembly routine gives up control.

If the timer runs out, the all reassembly resources for this buffer identifier are released. The initial setting of the timer is a lower bound on the reassembly waiting time. This is because the waiting time will be increased if the Time to Live in the arriving fragment is greater than the current timer value but will not be decreased if it is less. The maximum this timer value could reach is the maximum time to live (approximately 4.25 minutes). The current recommendation for the initial timer setting is 15 seconds. This may be changed as experience with

September 1981

this protocol accumulates. Note that the choice of this parameter value is related to the buffer capacity available and the data rate of the transmission medium; that is, data rate times timer value equals buffer size (e.g., 10Kb/s X 15s = 150Kb).

Notation:

FO - Fragment Offset
IHL - Internet Header Length
MF - More Fragments flag
TTL - Time To Live
NFB - Number of Fragment Blocks
TL - Total Length
TDL - Total Data Length
BUFID - Buffer Identifier
RCVBT - Fragment Received Bit Table
TLB - Timer Lower Bound

Procedure:

```
(1)  BUFID <- source|destination|protocol|identification;
(2)  IF FO = 0 AND MF = 0
(3)    THEN IF buffer with BUFID is allocated
(4)      THEN flush all reassembly for this BUFID;
(5)      Submit datagram to next step; DONE.
(6)  ELSE IF no buffer with BUFID is allocated
(7)    THEN allocate reassembly resources
        with BUFID;
        TIMER <- TLB; TDL <- 0;
(8)    put data from fragment into data buffer with
        BUFID from octet FO*8 to
        octet (TL-(IHL*4))+FO*8;
(9)    set RCVBT bits from FO
        to FO+((TL-(IHL*4)+7)/8);
(10)   IF MF = 0 THEN TDL <- TL-(IHL*4)+(FO*8)
(11)   IF FO = 0 THEN put header in header buffer
(12)   IF TDL # 0
(13)     AND all RCVBT bits from 0
        to (TDL+7)/8 are set
(14)     THEN TL <- TDL+(IHL*4)
(15)     Submit datagram to next step;
(16)     free all reassembly resources
        for this BUFID; DONE.
(17)   TIMER <- MAX(TIMER,TTL);
(18)   give up until next fragment or timer expires;
(19) timer expires: flush all reassembly with this BUFID; DONE.
```

In the case that two or more fragments contain the same data

either identically or through a partial overlap, this procedure will use the more recently arrived copy in the data buffer and datagram delivered.

Identification

The choice of the Identifier for a datagram is based on the need to provide a way to uniquely identify the fragments of a particular datagram. The protocol module assembling fragments judges fragments to belong to the same datagram if they have the same source, destination, protocol, and Identifier. Thus, the sender must choose the Identifier to be unique for this source, destination pair and protocol for the time the datagram (or any fragment of it) could be alive in the internet.

It seems then that a sending protocol module needs to keep a table of Identifiers, one entry for each destination it has communicated with in the last maximum packet lifetime for the internet.

However, since the Identifier field allows 65,536 different values, some host may be able to simply use unique identifiers independent of destination.

It is appropriate for some higher level protocols to choose the identifier. For example, TCP protocol modules may retransmit an identical TCP segment, and the probability for correct reception would be enhanced if the retransmission carried the same identifier as the original transmission since fragments of either datagram could be used to construct a correct TCP segment.

Type of Service

The type of service (TOS) is for internet service quality selection. The type of service is specified along the abstract parameters precedence, delay, throughput, and reliability. These abstract parameters are to be mapped into the actual service parameters of the particular networks the datagram traverses.

Precedence. An independent measure of the importance of this datagram.

Delay. Prompt delivery is important for datagrams with this indication.

Throughput. High data rate is important for datagrams with this indication.

September 1981

Reliability. A higher level of effort to ensure delivery is important for datagrams with this indication.

For example, the ARPANET has a priority bit, and a choice between "standard" messages (type 0) and "uncontrolled" messages (type 3). (the choice between single packet and multipacket messages can also be considered a service parameter). The uncontrolled messages tend to be less reliably delivered and suffer less delay. Suppose an internet datagram is to be sent through the ARPANET. Let the internet type of service be given as:

Precedence:	5
Delay:	0
Throughput:	1
Reliability:	1

In this example, the mapping of these parameters to those available for the ARPANET would be to set the ARPANET priority bit on since the Internet precedence is in the upper half of its range, to select standard messages since the throughput and reliability requirements are indicated and delay is not. More details are given on service mappings in "Service Mappings" [8].

Time to Live

The time to live is set by the sender to the maximum time the datagram is allowed to be in the internet system. If the datagram is in the internet system longer than the time to live, then the datagram must be destroyed.

This field must be decreased at each point that the internet header is processed to reflect the time spent processing the datagram. Even if no local information is available on the time actually spent, the field must be decremented by 1. The time is measured in units of seconds (i.e. the value 1 means one second). Thus, the maximum time to live is 255 seconds or 4.25 minutes. Since every module that processes a datagram must decrease the TTL by at least one even if it processes the datagram in less than a second, the TTL must be thought of only as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded, and to bound the maximum datagram lifetime.

Some higher level reliable connection protocols are based on assumptions that old duplicate datagrams will not arrive after a certain time elapses. The TTL is a way for such protocols to have an assurance that their assumption is met.

Options

The options are optional in each datagram, but required in implementations. That is, the presence or absence of an option is the choice of the sender, but each internet module must be able to parse every option. There can be several options present in the option field.

The options might not end on a 32-bit boundary. The internet header must be filled out with octets of zeros. The first of these would be interpreted as the end-of-options option, and the remainder as internet header padding.

Every internet module must be able to act on every option. The Security Option is required if classified, restricted, or compartmented traffic is to be passed.

Checksum

The internet header checksum is recomputed if the internet header is changed. For example, a reduction of the time to live, additions or changes to internet options, or due to fragmentation. This checksum at the internet level is intended to protect the internet header fields from transmission errors.

There are some applications where a few data bit errors are acceptable while retransmission delays are not. If the internet protocol enforced data correctness such applications could not be supported.

Errors

Internet protocol errors may be reported via the ICMP messages [3].

3.3. Interfaces

The functional description of user interfaces to the IP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different IP implementations may have different user interfaces. However, all IPs must provide a certain minimum set of services to guarantee that all IP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all IP implementations.

Internet protocol interfaces on one side to the local network and on the other side to either a higher level protocol or an application program. In the following, the higher level protocol or application

September 1981

program (or even a gateway program) will be called the "user" since it is using the internet module. Since internet protocol is a datagram protocol, there is minimal memory or state maintained between datagram transmissions, and each call on the internet protocol module by the user supplies all information necessary for the IP to perform the service requested.

An Example Upper Level Interface

The following two example calls satisfy the requirements for the user to internet protocol module communication ("=>" means returns):

SEND (src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt => result)

where:

- src = source address
- dst = destination address
- prot = protocol
- TOS = type of service
- TTL = time to live
- BufPTR = buffer pointer
- len = length of buffer
- Id = Identifier
- DF = Don't Fragment
- opt = option data
- result = response
 - OK = datagram sent ok
 - Error = error in arguments or local error

Note that the precedence is included in the TOS and the security/compartments is passed as an option.

RECV (BufPTR, prot, => result, src, dst, TOS, len, opt)

where:

- BufPTR = buffer pointer
- prot = protocol
- result = response
 - OK = datagram received ok
 - Error = error in arguments
- len = length of buffer
- src = source address
- dst = destination address
- TOS = type of service
- opt = option data

September 1981

Internet Protocol
Specification

When the user sends a datagram, it executes the SEND call supplying all the arguments. The internet protocol module, on receiving this call, checks the arguments and prepares and sends the message. If the arguments are good and the datagram is accepted by the local network, the call returns successfully. If either the arguments are bad, or the datagram is not accepted by the local network, the call returns unsuccessfully. On unsuccessful returns, a reasonable report must be made as to the cause of the problem, but the details of such reports are up to individual implementations.

When a datagram arrives at the internet protocol module from the local network, either there is a pending RECV call from the user addressed or there is not. In the first case, the pending call is satisfied by passing the information from the datagram to the user. In the second case, the user addressed is notified of a pending datagram. If the user addressed does not exist, an ICMP error message is returned to the sender, and the data is discarded.

The notification of a user may be via a pseudo interrupt or similar mechanism, as appropriate in the particular operating system environment of the implementation.

A user's RECV call may then either be immediately satisfied by a pending datagram, or the call may be pending until a datagram arrives.

The source address is included in the send call in case the sending host has several addresses (multiple physical connections or logical addresses). The internet module must check to see that the source address is one of the legal address for this host.

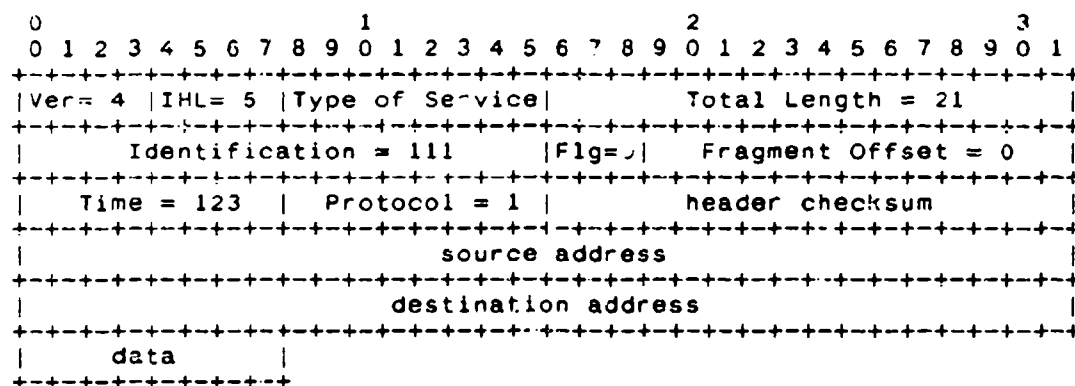
An implementation may also allow or require a call to the internet module to indicate interest in or reserve exclusive use of a class of datagrams (e.g., all those with a certain value in the protocol field).

This section functionally characterizes a USER/IP interface. The notation used is similar to most procedure of function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs), or any other form of interprocess communication.

APPENDIX A: Examples & Scenarios

Example 1:

This is an example of the minimal data carrying internet datagram:



Example Internet Datagram

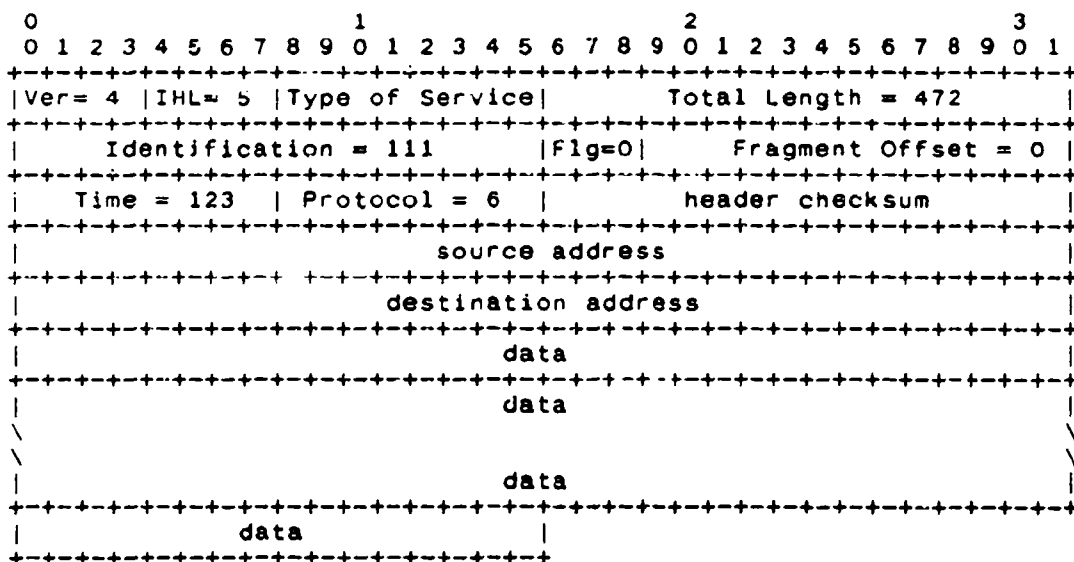
Figure 5.

Note that each tick mark represents one bit position.

This is a internet datagram in version 4 of internet protocol; the internet header consists of five 32 bit words, and the total length of the datagram is 21 octets. This datagram is a complete datagram (not a fragment).

Example 2:

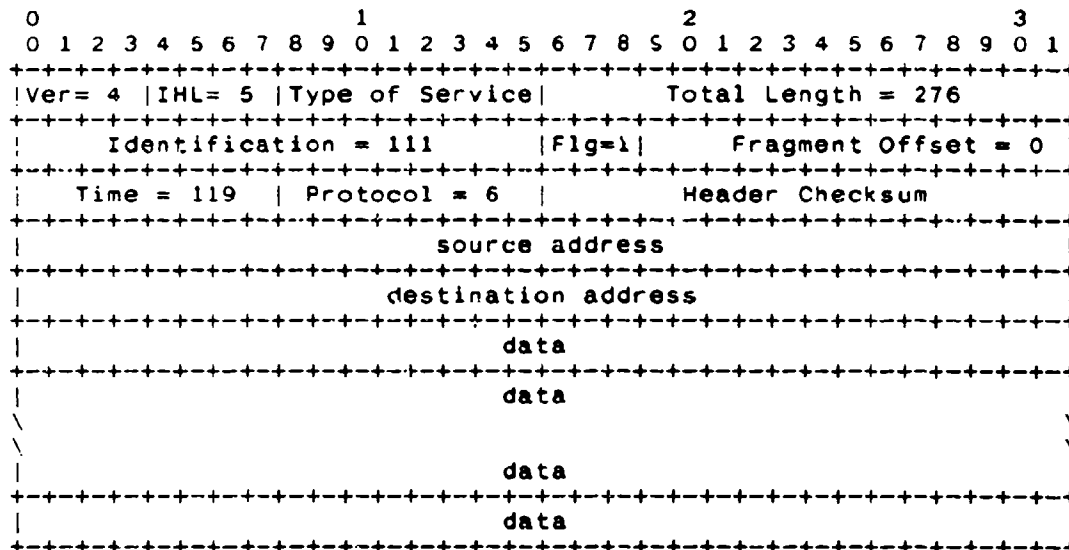
In this example, we show first a moderate size internet datagram (452 data octets), then two internet fragments that might result from the fragmentation of this datagram if the maximum sized transmission allowed were 280 octets.



Example Internet Datagram

Figure 6.

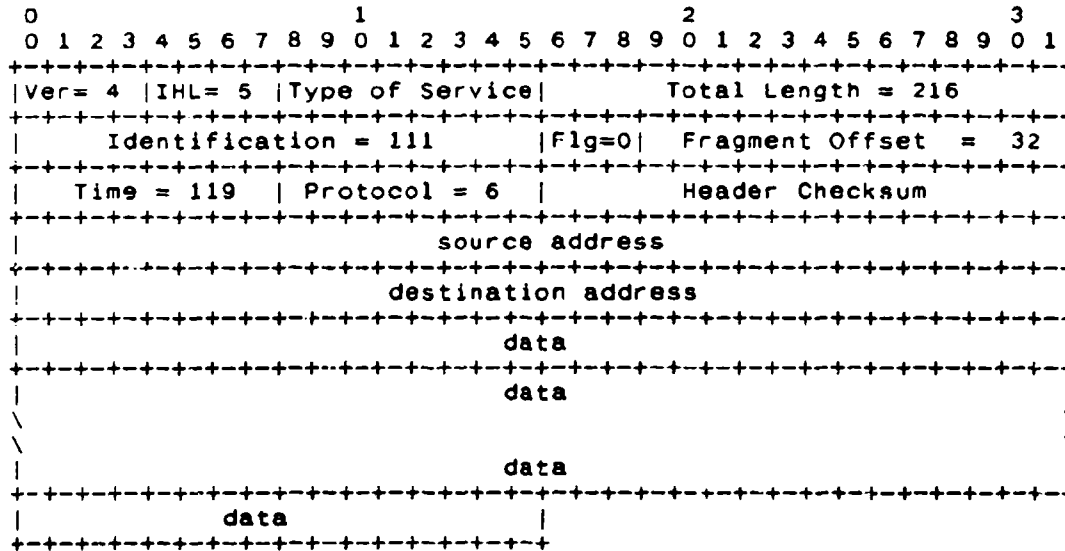
Now the first fragment that results from splitting the datagram after 256 data octets.



Example Internet Fragment

Figure 7.

And the second fragment.



Example Internet Fragment

Figure 8.

Example 3:

Here, we show an example of a datagram containing options:

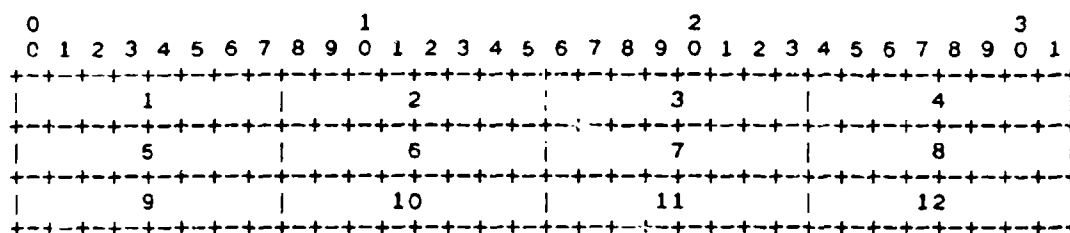
0					1					2					3						
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Ver= 4					IHL= 8					Type of Service					Total Length = 576						
Identification = 111										Flg=0					Fragment Offset = 0						
Time = 123					Protocol = 6					Header Checksum											
source address																					
destination address																					
Opt. Code = x					Opt. Len.= 3					option value					Opt. Code = x						
Opt. Len. = 4					option value					Opt. Code = 1											
Opt. Code = y					Opt. Len. = 3					option value					Opt. Code = 0						
data																					
data																					
data																					

Example Internet Datagram

Figure 9.

APPENDIX B: Data Transmission Order

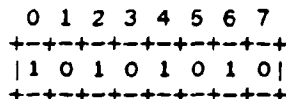
The order of transmission of the header and data described in this document is resolved to the octet level. Whenever a diagram shows a group of octets, the order of transmission of those octets is the normal order in which they are read in English. For example, in the following diagram the octets are transmitted in the order they are numbered.



Transmission Order of Bytes

Figure 10.

Whenever an octet represents a numeric quantity the left most bit in the diagram is the high order or most significant bit. That is, the bit labeled 0 is the most significant bit. For example, the following diagram represents the value 170 (decimal).



Significance of Bits

Figure 11.

Similarly, whenever a multi-octet field represents a numeric quantity the left most bit of the whole field is the most significant bit. When a multi-octet quantity is transmitted the most significant octet is transmitted first.

Internet Protocol

September 1981

[Page 40]

(96)

GLOSSARY

1822

BBN Report 1822, "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ARPANET leader

The control information on an ARPANET message at the host-IMP interface.

ARPANET message

The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet

A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

Destination

The destination address, an internet header field.

DF

The Don't Fragment bit carried in the flags field.

Flags

An internet header field carrying various control flags.

Fragment Offset

This internet header field indicates where in the internet datagram a fragment belongs.

GGP

Gateway to Gateway Protocol, the protocol used primarily between gateways to control routing and other gateway functions.

header

Control information at the beginning of a message, segment, datagram, packet or block of data.

ICMP

Internet Control Message Protocol, implemented in the internet module, the ICMP is used from gateways to hosts and between hosts to report errors and make routing suggestions.

September 1981

Internet Protocol
Glossary

Identification

An internet header field carrying the identifying value assigned by the sender to aid in assembling the fragments of a datagram.

IHL

The internet header field Internet Header Length is the length of the internet header measured in 32 bit words.

IMP

The Interface Message Processor, the packet switch of the ARPANET.

Internet Address

A four octet (32 bit) source or destination address consisting of a Network field and a Local Address field.

internet datagram

The unit of data exchanged between a pair of internet modules (includes the internet header).

internet fragment

A portion of the data of an internet datagram with an internet header.

Local Address

The address of a host within a network. The actual mapping of an internet local address on to the host addresses in a network is quite general, allowing for many to one mappings.

MF

The More-Fragments Flag carried in the internet header flags field.

module

An implementation, usually in software, of a protocol or other procedure.

more-fragments flag

A flag indicating whether or not this internet datagram contains the end of an internet datagram, carried in the internet header Flags field.

NFB

The Number of Fragment Blocks in a the data portion of an internet fragment. That is, the length of a portion of data measured in 8 octet units.

September 1981

Internet Protocol
Glossary

octet An eight bit byte.

Options The internet header Options field may contain several options, and each option may be several octets in length.

Padding The internet header Padding field is used to ensure that the data begins on 32 bit word boundary. The padding is zero.

Protocol In this document, the next higher level protocol identifier, an internet header field.

Rest The local address portion of an Internet Address.

Source The source address, an internet header field.

TCP Transmission Control Protocol: A host-to-host protocol for reliable communication in internet environments.

TCP Segment The unit of data exchanged between TCP modules (including the TCP header).

TFTP Trivial File Transfer Protocol: A simple file transfer protocol built on UDP.

Time to Live An internet header field which indicates the upper bound on how long this Internet datagram may exist.

TOS Type of Service

Total Length The internet header field Total Length is the length of the datagram in octets including internet header and data.

TTL Time to Live

September 1981

Internet Protocol
Glossary

Type of Service

An internet header field which indicates the type (or quality) of service for this internet datagram.

UDP

User Datagram Protocol: A user level protocol for transaction oriented applications.

User

The user of the internet protocol. This may be a higher level protocol module, an application program, or a gateway program.

Version

The Version field indicates the format of the internet header.

RFC-792

INTERNET CONTROL MESSAGE PROTOCOL

September 1981

(101)

Network Working Group
Request for Comments: 792

Updates: RFCs 777, 760
Updates: IENs 109, 128

J. Postel
ISI
September 1981

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols that use IP must implement their own reliability procedures if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. Also ICMP messages are only sent about errors in handling fragment zero of fragmented datagrams. (Fragment zero has the fragment offset equal zero).

Message Formats

ICMP messages are sent using the basic IP header. The first octet of the data portion of the datagram is a ICMP type field; the value of this field determines the format of the remaining data. Any field labeled "unused" is reserved for later extensions and must be zero when sent, but receivers should not use these fields (except to include them in the checksum). Unless otherwise noted under the individual format descriptions, the values of the internet header fields are as follows:

Version

4

IHL

Internet header length in 32-bit words.

Type of Service

0

Total Length

Length of internet header and data in octets.

Identification, Flags, Fragment Offset

Used in fragmentation, see [1].

Time to Live

Time to live in seconds; as this field is decremented at each machine in which the datagram is processed, the value in this field should be at least as great as the number of gateways which this datagram will traverse.

Protocol

ICMP = 1

Header Checksum

The 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

September 1981
RFC 792

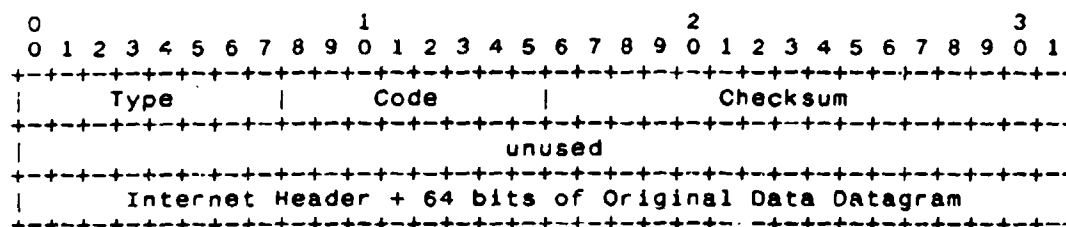
Source Address

The address of the gateway or host that composes the ICMP message.
Unless otherwise noted, this can be any of a gateway's addresses.

Destination Address

The address of the gateway or host to which the message should be sent.

Destination Unreachable Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

3

Code

- 0 = net unreachable;
- 1 = host unreachable;
- 2 = protocol unreachable;
- 3 = port unreachable;
- 4 = fragmentation needed and DF set;
- 5 = source route failed.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original

September 1981
RFC 792

datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

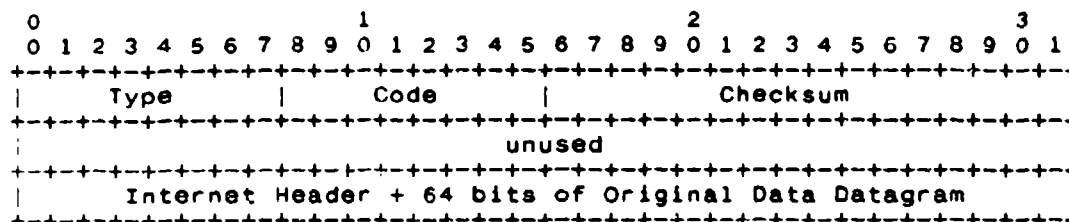
If, according to the information in the gateway's routing tables, the network specified in the internet destination field of a datagram is unreachable, e.g., the distance to the network is infinity, the gateway may send a destination unreachable message to the internet source host of the datagram. In addition, in some networks, the gateway may be able to determine if the internet destination host is unreachable. Gateways in these networks may send destination unreachable messages to the source host when the destination host is unreachable.

If, in the destination host, the IP module cannot deliver the datagram because the indicated protocol module or process port is not active, the destination host may send a destination unreachable message to the source host.

Another case is when a datagram must be fragmented to be forwarded by a gateway yet the Don't Fragment flag is on. In this case the gateway must discard the datagram and may return a destination unreachable message.

Codes 0, 1, 4, and 5 may be received from a gateway. Codes 2 and 3 may be received from a host.

Time Exceeded Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

11

Code

0 = time to live exceeded in transit;

1 = fragment reassembly time exceeded.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

If the gateway processing a datagram finds the time to live field

September 1981
RFC 792

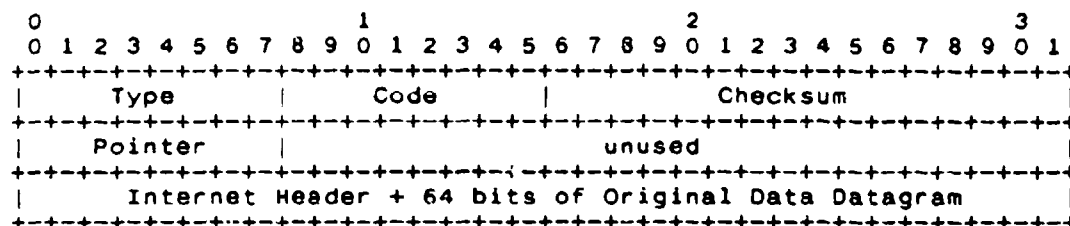
is zero it must discard the datagram. The gateway may also notify the source host via the time exceeded message.

If a host reassembling a fragmented datagram cannot complete the reassembly due to missing fragments within its time limit it discards the datagram, and it may send a time exceeded message.

If fragment zero is not available then no time exceeded need be sent at all.

Code 0 may be received from a gateway. Code 1 may be received from a host.

Parameter Problem Message



IP Fields:

Destination Address

The source network and address from the original datagram's data.

ICMP Fields:

Type

12

Code

0 = pointer indicates the error.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Pointer

If code = 0, identifies the octet where an error was detected.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

September 1981
RFC 792

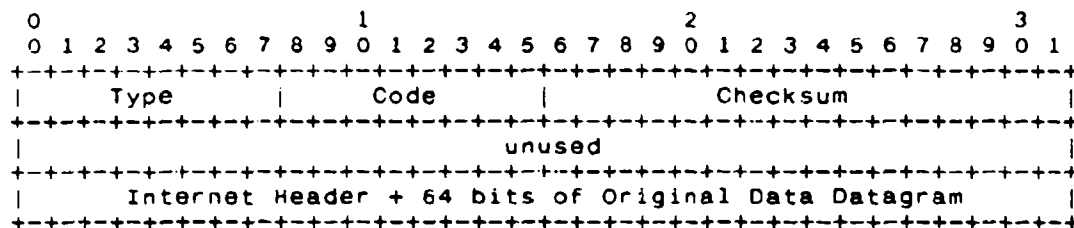
Description

If the gateway or host processing a datagram finds a problem with the header parameters such that it cannot complete processing the datagram it must discard the datagram. One potential source of such a problem is with incorrect arguments in an option. The gateway or host may also notify the source host via the parameter problem message. This message is only sent if the error caused the datagram to be discarded.

The pointer identifies the octet of the original datagram's header where the error was detected (it may be in the middle of an option). For example, 1 indicates something is wrong with the Type of Service, and (if there are options present) 20 indicates something is wrong with the type code of the first option.

Code 0 may be received from a gateway or a host.

Source Quench Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

4

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

Description

A gateway may discard internet datagrams if it does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network. If a gateway

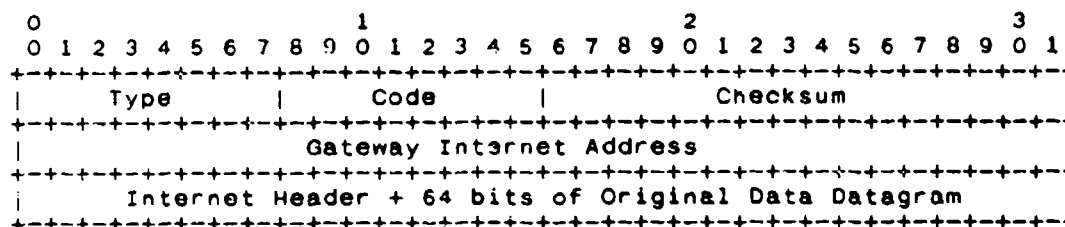
September 1981
RFC 792

discards a datagram, it may send a source quench message to the internet source host of the datagram. A destination host may also send a source quench message if datagrams arrive too fast to be processed. The source quench message is a request to the host to cut back the rate at which it is sending traffic to the internet destination. The gateway may send a source quench message for every message that it discards. On receipt of a source quench message, the source host should cut back the rate at which it is sending traffic to the specified destination until it no longer receives source quench messages from the gateway. The source host can then gradually increase the rate at which it sends traffic to the destination until it again receives source quench messages.

The gateway or host may send the source quench message when it approaches its capacity limit rather than waiting until the capacity is exceeded. This means that the data datagram which triggered the source quench message may be delivered.

Code 0 may be received from a gateway or a host.

Redirect Message



IP Fields:

Destination Address

The source network and address of the original datagram's data.

ICMP Fields:

Type

5

Code

0 = Redirect datagrams for the Network.

1 = Redirect datagrams for the Host.

2 = Redirect datagrams for the Type of Service and Network.

3 = Redirect datagrams for the Type of Service and Host.

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Gateway Internet Address

Address of the gateway to which traffic for the network specified in the internet destination network field of the original datagram's data should be sent.

Internet Header + 64 bits of Data Datagram

The internet header plus the first 64 bits of the original datagram's data. This data is used by the host to match the message to the appropriate process. If a higher level protocol uses port numbers, they are assumed to be in the first 64 data bits of the original datagram's data.

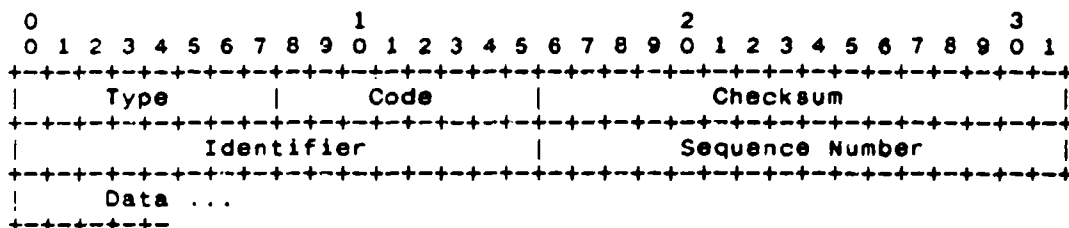
Description

The gateway sends a redirect message to a host in the following situation. A gateway, G1, receives an internet datagram from a host on a network to which the gateway is attached. The gateway, G1, checks its routing table and obtains the address of the next gateway, G2, on the route to the datagram's internet destination network, X. If G2 and the host identified by the internet source address of the datagram are on the same network, a redirect message is sent to the host. The redirect message advises the host to send its traffic for network X directly to gateway G2 as this is a shorter path to the destination. The gateway forwards the original datagram's data to its internet destination.

For datagrams with the IP source route options and the gateway address in the destination address field, a redirect message is not sent even if there is a better route to the ultimate destination than the next address in the source route.

Codes 0, 1, 2, and 3 may be received from a gateway.

Echo or Echo Reply Message



IP Fields:

Addresses

The address of the source in an echo message will be the destination of the echo reply message. To form an echo reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum recomputed.

IP Fields:

Type

8 for echo message;

0 for echo reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. If the total length is odd, the received data is padded with one octet of zeros for computing the checksum. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching echos and replies, may be zero.

Sequence Number

September 1981
RFC 792

If code = 0, a sequence number to aid in matching echos and replies, may be zero.

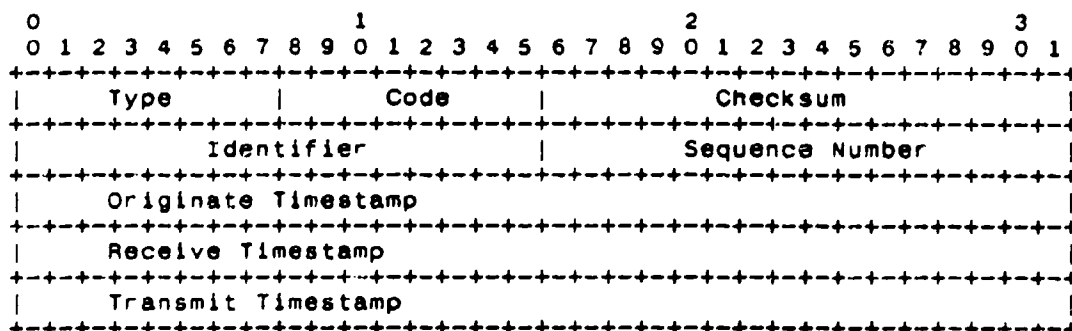
Description

The data received in the echo message must be returned in the echo reply message.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the echo requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each echo request sent. The echoer returns these same values in the echo reply.

Code 0 may be received from a gateway or a host.

Timestamp or Timestamp Reply Message



IP Fields:

Addresses

The address of the source in a timestamp message will be the destination of the timestamp reply message. To form a timestamp reply message, the source and destination addresses are simply reversed, the type code changed to 14, and the checksum recomputed.

IP Fields:

Type

- 13 for timestamp message;
- 14 for timestamp reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching timestamp and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching timestamp and replies, may be zero.

Description

The data received (a timestamp) in the message is returned in the reply together with an additional timestamp. The timestamp is 32 bits of milliseconds since midnight UT. One use of these timestamps is described by Mills [5].

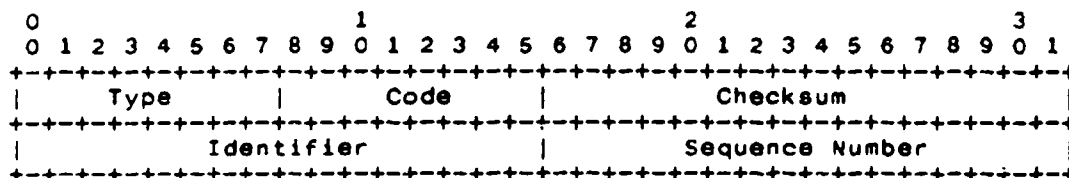
The Originate Timestamp is the time the sender last touched the message before sending it, the Receive Timestamp is the time the echoer first touched it on receipt, and the Transmit Timestamp is the time the echoer last touched the message on sending it.

If the time is not available in milliseconds or cannot be provided with respect to midnight UT then any time can be inserted in a timestamp provided the high order bit of the timestamp is also set to indicate this non-standard value.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Information Request or Information Reply Message



IP Fields:

Addresses

The address of the source in a information request message will be the destination of the information reply message. To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed.

IP Fields:

Type

- 15 for information request message;
- 16 for information reply message.

Code

0

Checksum

The checksum is the 16-bit ones's complement of the one's complement sum of the ICMP message starting with the ICMP Type. For computing the checksum, the checksum field should be zero. This checksum may be replaced in the future.

Identifier

If code = 0, an identifier to aid in matching request and replies, may be zero.

Sequence Number

If code = 0, a sequence number to aid in matching request and replies, may be zero.

September 1981
RFC 792

Description

This message may be sent with the source network in the IP header source and destination address fields zero (which means "this" network). The replying IP module should send the reply with the addresses fully specified. This message is a way for a host to find out the number of the network it is on.

The identifier and sequence number may be used by the echo sender to aid in matching the replies with the requests. For example, the identifier might be used like a port in TCP or UDP to identify a session, and the sequence number might be incremented on each request sent. The destination returns these same values in the reply.

Code 0 may be received from a gateway or a host.

Summary of Message Types

- 0 Echo Reply
- 3 Destination Unreachable
- 4 Source Quench
- 5 Redirect
- 8 Echo
- 11 Time Exceeded
- 12 Parameter Problem
- 13 Timestamp
- 14 Timestamp Reply
- 15 Information Request
- 16 Information Reply

September 1981
RFC 792

References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.
- [2] Cerf, V., "The Catenet Model for Internetworking," IEN 48, Information Processing Techniques Office, Defense Advanced Research Projects Agency, July 1978.
- [3] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Beranek and Newman, April 1979.
- [4] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Beranek and Newman, August 1979.
- [5] Mills, D., "DCNET Internet Clock Service," RFC 778, COMSAT Laboratories, April 1981.

HOST LEVEL

USER DATAGRAM PROTOCOL

TRANSMISSION CONTROL PROTOCOL

RFC-768

USER DATAGRAM PROTOCOL

28 August 1980

(125)

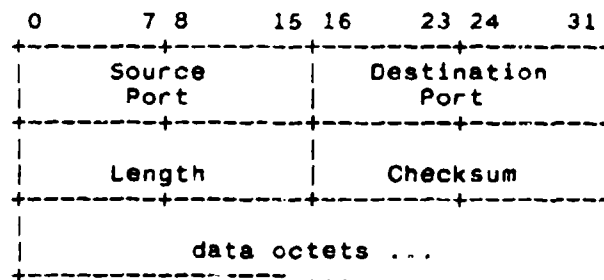
USER DATAGRAM PROTOCOL

Introduction

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

Format



User Datagram Header Format

Fields

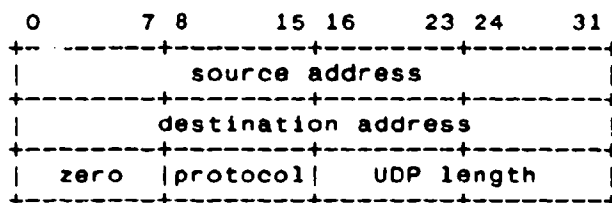
Source Port is an optional field. when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

Destination Port has a meaning within the context of a particular internet destination address.

Length is the length in octets of this user datagram including this header and the data. (This means the minimum value of the length is eight.)

Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

The pseudo header conceptually prefixed to the UDP header contains the source address, the destination address, the protocol, and the UDP length. This information gives protection against misrouted datagrams. This checksum procedure is the same as is used in TCP.



If the computed checksum is zero, it is transmitted as all ones (the equivalent in one's complement arithmetic). An all zero transmitted checksum value means that the transmitter generated no checksum (for debugging or for higher level protocols that don't care).

User Interface

A user interface should allow

the creation of new receive ports.

receive operations on the receive ports that return the data octets and an indication of source port and source address,

and an operation that allows a datagram to be sent, specifying the data, source and destination ports and addresses to be sent.

28 Aug 1980
RFC 768

User Datagram Protocol
IP Interface

IP Interface

The UDP module must be able to determine the source and destination internet addresses and the protocol field from the internet header. One possible UDP/IP interface would return the whole internet datagram including all of the internet header in response to a receive operation. Such an interface would also allow the UDP to pass a full internet datagram complete with header to the IP to send. The IP would verify certain fields for consistency and compute the internet header checksum.

Protocol Application

The major uses of this protocol is the Internet Name Server [3], and the Trivial File Transfer [4].

Protocol Number

This is protocol 17 (21 octal) when used in the Internet Protocol. Other protocol numbers are listed in [5].

References

- [1] Postel, J., "Internet Protocol," RFC 760, USC/Information Sciences Institute, January 1980.
- [2] Postel, J., "Transmission Control Protocol," RFC 761, USC/Information Sciences Institute, January 1980.
- [3] Postel, J., "Internet Name Server," USC/Information Sciences Institute, IEN 116, August 1979.
- [4] Sollins, K., "The TFTP Protocol," Massachusetts Institute of Technology, IEN 133, January 1980.
- [5] Postel, J., "Assigned Numbers," USC/Information Sciences Institute, RFC 762, January 1980.

RFC-793

TRANSMISSION CONTROL PROTOCOL

September 1981

(131)

TABLE OF CONTENTS

PREFACE	11
1. INTRODUCTION	1
1.1 Motivation	1
1.2 Scope	2
1.3 About This Document	2
1.4 Interfaces	3
1.5 Operation	3
2. PHILOSOPHY	7
2.1 Elements of the Internetwork System	7
2.2 Model of Operation	7
2.3 The Host Environment	8
2.4 Interfaces	9
2.5 Relation to Other Protocols	9
2.6 Reliable Communication	9
2.7 Connection Establishment and Clearing	10
2.8 Data Communication	12
2.9 Precedence and Security	13
2.10 Robustness Principle	13
3. FUNCTIONAL SPECIFICATION	15
3.1 Header Format	15
3.2 Terminology	19
3.3 Sequence Numbers	24
3.4 Establishing a connection	30
3.5 Closing a Connection	37
3.6 Precedence and Security	40
3.7 Data Communication	40
3.8 Interfaces	44
3.9 Event Processing	52
GLOSSARY	79
REFERENCES	85

September 1981

Transmission Control Protocol

PREFACE

This document describes a revised version of the DoD Standard Transmission Control Protocol (TCP). There have been nine earlier editions of the ARPA TCP specification on which this standard is based, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition clarifies several details and removes the end-of-letter buffer-size adjustments, and redescribes the letter mechanism as a push function.

Jon Postel

Editor

RFC: 793
Replaces: RFC 761
IENs: 129, 124, 112, 81,
55, 44, 40, 27, 21, 5

TRANSMISSION CONTROL PROTOCOL

DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION

1. INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

1.1. Motivation

Computer communication systems are playing an increasingly important role in military, government, and civilian environments. This document focuses its attention primarily on military computer communication requirements, especially robustness in the presence of communication unreliability and availability in the presence of congestion, but many of these problems are found in the civilian and government sector as well.

As strategic and tactical computer communication networks are developed and deployed, it is essential to provide means of interconnecting them and to provide standard interprocess communication protocols which can support a broad range of applications. In anticipation of the need for such standards, the Deputy Undersecretary of Defense for Research and Engineering has declared the Transmission Control Protocol (TCP) described herein to be a basis for DoD-wide inter-process communication protocol standardization.

TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks. Very few assumptions are made as to the reliability of the communication protocols below the TCP layer. TCP assumes it can obtain a simple, potentially unreliable datagram service from the lower level protocols. In principle, the TCP should be able to operate above a wide spectrum of communication systems ranging from hard-wired connections to packet-switched or circuit-switched networks.

September 1981

TCP is based on concepts first described by Cerf and Kahn in [1]. The TCP fits into a layered protocol architecture just above a basic Internet Protocol [2] which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

Protocol Layering

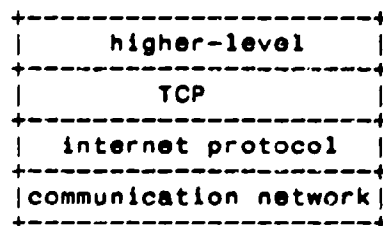


Figure 1

Much of this document is written in the context of TCP implementations which are co-resident with higher level protocols in the host computer. Some computer systems will be connected to networks via front-end computers which house the TCP and internet protocol layers, as well as network specific software. The TCP specification describes an interface to the higher level protocols which appears to be implementable even for the front-end case, as long as a suitable host-to-front end protocol is implemented.

1.2. Scope

The TCP is intended to provide a reliable process-to-process communication service in a multinet environment. The TCP is intended to be a host-to-host protocol in common use in multiple networks.

1.3. About this Document

This document represents a specification of the behavior required of any TCP implementation, both in its interactions with higher level protocols and in its interactions with other TCPs. The rest of this

section offers a very brief view of the protocol interfaces and operation. Section 2 summarizes the philosophical basis for the TCP design. Section 3 offers both a detailed description of the actions required of TCP when various events occur (arrival of new segments, user calls, errors, etc.) and the details of the formats of TCP segments.

1.4. Interfaces

The TCP interfaces on one side to user or application processes and on the other side to a lower level protocol such as Internet Protocol.

The interface between an application process and the TCP is illustrated in reasonable detail. This interface consists of a set of calls much like the calls an operating system provides to an application process for manipulating files. For example, there are calls to open and close connections and to send and receive data on established connections. It is also expected that the TCP can asynchronously communicate with application programs. Although considerable freedom is permitted to TCP implementors to design interfaces which are appropriate to a particular operating system environment, a minimum functionality is required at the TCP/user interface for any valid implementation.

The interface between TCP and lower level protocol is essentially unspecified except that it is assumed there is a mechanism whereby the two levels can asynchronously pass information to each other. Typically, one expects the lower level protocol to specify this interface. TCP is designed to work in a very general environment of interconnected networks. The lower level protocol which is assumed throughout this document is the Internet Protocol [2].

1.5. Operation

As noted above, the primary purpose of the TCP is to provide reliable, securable logical circuit or connection service between pairs of processes. To provide this service on top of a less reliable internet communication system requires facilities in the following areas:

- Basic Data Transfer
- Reliability
- Flow Control
- Multiplexing
- Connections
- Precedence and Security

The basic operation of the TCP in each of these areas is described in the following paragraphs.

September 1981

Basic Data Transfer:

The TCP is able to transfer a continuous stream of octets in each direction between its users by packaging some number of octets into segments for transmission through the internet system. In general, the TCPs decide when to block and forward data at their own convenience.

Sometimes users need to be sure that all the data they have submitted to the TCP has been transmitted. For this purpose a push function is defined. To assure that data submitted to a TCP is actually transmitted the sending user indicates that it should be pushed through to the receiving user. A push causes the TCPs to promptly forward and deliver data up to that point to the receiver. The exact push point might not be visible to the receiving user and the push function does not supply a record boundary marker.

Reliability:

The TCP must recover from data that is damaged, lost, duplicated, or delivered out of order by the internet communication system. This is achieved by assigning a sequence number to each octet transmitted, and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. At the receiver, the sequence numbers are used to correctly order segments that may be received out of order and to eliminate duplicates. Damage is handled by adding a checksum to each segment transmitted, checking it at the receiver, and discarding damaged segments.

As long as the TCPs continue to function properly and the internet system does not become completely partitioned, no transmission errors will affect the correct delivery of data. TCP recovers from internet communication system errors.

Flow Control:

TCP provides a means for the receiver to govern the amount of data sent by the sender. This is achieved by returning a "window" with every ACK indicating a range of acceptable sequence numbers beyond the last segment successfully received. The window indicates an allowed number of octets that the sender may transmit before receiving further permission.

September 1981

Transmission Control Protocol
Introduction

Multiplexing:

To allow for many processes within a single Host to use TCP communication facilities simultaneously, the TCP provides a set of addresses or ports within each host. Concatenated with the network and host addresses from the internet communication layer, this forms a socket. A pair of sockets uniquely identifies each connection. That is, a socket may be simultaneously used in multiple connections.

The binding of ports to processes is handled independently by each Host. However, it proves useful to attach frequently used processes (e.g., a "logger" or timesharing service) to fixed sockets which are made known to the public. These services can then be accessed through the known addresses. Establishing and learning the port addresses of other processes may involve more dynamic mechanisms.

Connections:

The reliability and flow control mechanisms described above require that TCPs initialize and maintain certain status information for each data stream. The combination of this information, including sockets, sequence numbers, and window sizes, is called a connection. Each connection is uniquely specified by a pair of sockets identifying its two sides.

When two processes wish to communicate, their TCP's must first establish a connection (initialize the status information on each side). When their communication is complete, the connection is terminated or closed to free the resources for other uses.

Since connections must be established between unreliable hosts and over the unreliable internet communication system, a handshake mechanism with clock-based sequence numbers is used to avoid erroneous initialization of connections.

Precedence and Security:

The users of TCP may indicate the security and precedence of their communication. Provision is made for default values to be used when these features are not needed.

2. PHILOSOPHY

2.1. Elements of the Internetwork System

The internetwork environment consists of hosts connected to networks which are in turn interconnected via gateways. It is assumed here that the networks may be either local networks (e.g., the ETHERNET) or large networks (e.g., the ARPANET), but in any case are based on packet switching technology. The active agents that produce and consume messages are processes. Various levels of protocols in the networks, the gateways, and the hosts support an interprocess communication system that provides two-way data flow on logical connections between process ports.

The term packet is used generically here to mean the data of one transaction between a host and its network. The format of data blocks exchanged within the a network will generally not be of concern to us.

Hosts are computers attached to a network, and from the communication network's point of view, are the sources and destinations of packets. Processes are viewed as the active elements in host computers (in accordance with the fairly common definition of a process as a program in execution). Even terminals and files or other I/O devices are viewed as communicating with each other through the use of processes. Thus, all communication is viewed as inter-process communication.

Since a process may need to distinguish among several communication streams between itself and another process (or processes), we imagine that each process may have a number of ports through which it communicates with the ports of other processes.

2.2. Model of Operation

Processes transmit data by calling on the TCP and passing buffers of data as arguments. The TCP packages the data from these buffers into segments and calls on the internet module to transmit each segment to the destination TCP. The receiving TCP places the data from a segment into the receiving user's buffer and notifies the receiving user. The TCPs include control information in the segments which they use to ensure reliable ordered data transmission.

The model of internet communication is that there is an internet protocol module associated with each TCP which provides an interface to the local network. This internet module packages TCP segments inside internet datagrams and routes these datagrams to a destination internet module or intermediate gateway. To transmit the datagram through the local network, it is embedded in a local network packet.

The packet switches may perform further packaging, fragmentation, or

September 1981

other operations to achieve the delivery of the local packet to the destination internet module.

At a gateway between networks, the internet datagram is "unwrapped" from its local packet and examined to determine through which network the internet datagram should travel next. The internet datagram is then "wrapped" in a local packet suitable to the next network and routed to the next gateway, or to the final destination.

A gateway is permitted to break up an internet datagram into smaller internet datagram fragments if this is necessary for transmission through the next network. To do this, the gateway produces a set of internet datagrams; each carrying a fragment. Fragments may be further broken into smaller fragments at subsequent gateways. The internet datagram fragment format is designed so that the destination internet module can reassemble fragments into internet datagrams.

A destination internet module unwraps the segment from the datagram (after reassembling the datagram, if necessary) and passes it to the destination TCP.

This simple model of the operation glosses over many details. One important feature is the type of service. This provides information to the gateway (or internet module) to guide it in selecting the service parameters to be used in traversing the next network. Included in the type of service information is the precedence of the datagram. Datagrams may also carry security information to permit host and gateways that operate in multilevel secure environments to properly segregate datagrams for security considerations.

2.3. The Host Environment

The TCP is assumed to be a module in an operating system. The users access the TCP much like they would access the file system. The TCP may call on other operating system functions, for example, to manage data structures. The actual interface to the network is assumed to be controlled by a device driver module. The TCP does not call on the network device driver directly, but rather calls on the internet datagram protocol module which may in turn call on the device driver.

The mechanisms of TCP do not preclude implementation of the TCP in a front-end processor. However, in such an implementation, a host-to-front-end protocol must provide the functionality to support the type of TCP-user interface described in this document.

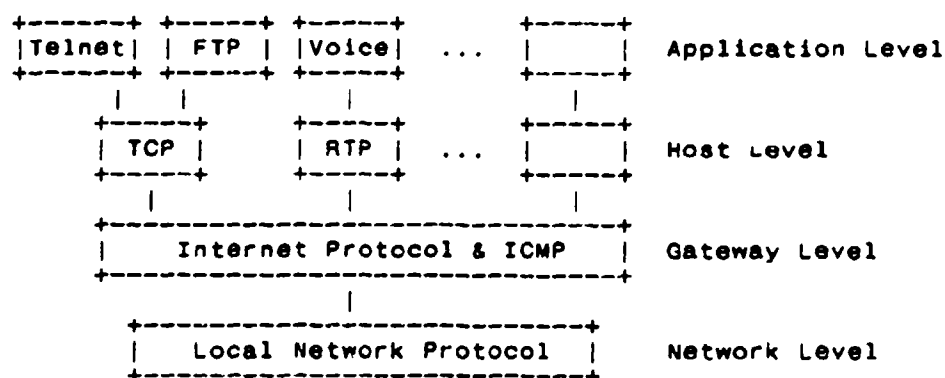
2.4. Interfaces

The TCP/user interface provides for calls made by the user on the TCP to OPEN or CLOSE a connection, to SEND or RECEIVE data, or to obtain STATUS about a connection. These calls are like other calls from user programs on the operating system, for example, the calls to open, read from, and close a file.

The TCP/internet interface provides calls to send and receive datagrams addressed to TCP modules in hosts anywhere in the internet system. These calls have parameters for passing the address, type of service, precedence, security, and other control information.

2.5. Relation to Other Protocols

The following diagram illustrates the place of the TCP in the protocol hierarchy:



Protocol Relationships

Figure 2.

It is expected that the TCP will be able to support higher level protocols efficiently. It should be easy to interface higher level protocols like the ARPANET Telnet or AUTODIN II THP to the TCP.

2.6. Reliable Communication

A stream of data sent on a TCP connection is delivered reliably and in order at the destination.

September 1981

Transmission Control Protocol Philosophy

Transmission is made reliable via the use of sequence numbers and acknowledgments. Conceptually, each octet of data is assigned a sequence number. The sequence number of the first octet of data in a segment is transmitted with that segment and is called the segment sequence number. Segments also carry an acknowledgment number which is the sequence number of the next expected data octet of transmissions in the reverse direction. When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts a timer; when the acknowledgment for that data is received, the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted.

An acknowledgment by TCP does not guarantee that the data has been delivered to the end user, but only that the receiving TCP has taken the responsibility to do so.

To govern the flow of data between TCPs, a flow control mechanism is employed. The receiving TCP reports a "window" to the sending TCP. This window specifies the number of octets, starting with the acknowledgment number, that the receiving TCP is currently prepared to receive.

2.7. Connection Establishment and Clearing

To identify the separate data streams that a TCP may handle, the TCP provides a port identifier. Since port identifiers are selected independently by each TCP they might not be unique. To provide for unique addresses within each TCP, we concatenate an internet address identifying the TCP with a port identifier to create a socket which will be unique throughout all networks connected together.

A connection is fully specified by the pair of sockets at the ends. A local socket may participate in many connections to different foreign sockets. A connection can be used to carry data in both directions, that is, it is "full duplex".

TCPs are free to associate ports with processes however they choose. However, several basic concepts are necessary in any implementation. There must be well-known sockets which the TCP associates only with the "appropriate" processes by some means. We envision that processes may "own" ports, and that processes can initiate connections only on the ports they own. (Means for implementing ownership is a local issue, but we envision a Request Port user command, or a method of uniquely allocating a group of ports to a given process, e.g., by associating the high order bits of a port name with a given process.)

A connection is specified in the OPEN call by the local port and foreign socket arguments. In return, the TCP supplies a (short) local

connection name by which the user refers to the connection in subsequent calls. There are several things that must be remembered about a connection. To store this information we imagine that there is a data structure called a Transmission Control Block (TCB). One implementation strategy would have the local connection name be a pointer to the TCB for this connection. The OPEN call also specifies whether the connection establishment is to be actively pursued, or to be passively waited for.

A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case a foreign socket of all zeros is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs.

A service process that wished to provide services for unknown other processes would issue a passive OPEN request with an unspecified foreign socket. Then a connection could be made with any process that requested a connection to this local socket. It would help if this local socket were known to be associated with this service.

Well-known sockets are a convenient mechanism for a priori associating a socket address with a standard service. For instance, the "Telnet-Server" process is permanently assigned to a particular socket, and other sockets are reserved for File Transfer, Remote Job Entry, text Generator, Echoer, and Sink processes (the last three being for test purposes). A socket address might be reserved for access to a "Look-Up" service which would return the specific socket at which a newly created service would be provided. The concept of a well-known socket is part of the TCP specification, but the assignment of sockets to services is outside this specification. (See [4].)

Processes can issue passive OPENs and wait for matching active OPENs from other processes and be informed by the TCP when connections have been established. Two processes which issue active OPENs to each other at the same time will be correctly connected. This flexibility is critical for the support of distributed computing in which components act asynchronously with respect to each other.

There are two principal cases for matching the sockets in the local passive OPENs and an foreign active OPENs. In the first case, the local passive OPENs has fully specified the foreign socket. In this case, the match must be exact. In the second case, the local passive OPENs has left the foreign socket unspecified. In this case, any foreign socket is acceptable as long as the local sockets match. Other possibilities include partially restricted matches.

September 1981

If there are several pending passive OPENs (recorded in TCBs) with the same local socket, an foreign active OPEN will be matched to a TCB with the specific foreign socket in the foreign active OPEN, if such a TCB exists, before selecting a TCB with an unspecified foreign socket.

The procedures to establish connections utilize the synchronize (SYN) control flag and involves an exchange of three messages. This exchange has been termed a three-way hand shake [3].

A connection is initiated by the rendezvous of an arriving segment containing a SYN and a waiting TCB entry each created by a user OPEN command. The matching of local and foreign sockets determines when a connection has been initiated. The connection becomes "established" when sequence numbers have been synchronized in both directions.

The clearing of a connection also involves the exchange of segments, in this case carrying the FIN control flag.

2.8. Data Communication

The data that flows on a connection may be thought of as a stream of octets. The sending user indicates in each SEND call whether the data in that call (and any preceeding calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.

A sending TCP is allowed to collect data from the sending user and to send that data in segments at its own convenience, until the push function is signaled, then it must send all unsent data. When a receiving TCP sees the PUSH flag, it must not wait for more data from the sending TCP before passing the data to the receiving process.

There is no necessary relationship between push functions and segment boundaries. The data in any particular segment may be the result of a single SEND call, in whole or part, or of multiple SEND calls.

The purpose of push function and the PUSH flag is to push data through from the sending user to the receiving user. It does not provide a record service.

There is a coupling between the push function and the use of buffers of data that cross the TCP/user interface. Each time a PUSH flag is associated with data placed into the receiving user's buffer, the buffer is returned to the user for processing even if the buffer is not filled. If data arrives that fills the user's buffer before a PUSH is seen, the data is passed to the user in buffer size units.

TCP also provides a means to communicate to the receiver of data that at some point further along in the data stream than the receiver is

September 1981

Transmission Control Protocol
Philosophy

currently reading there is urgent data. TCP does not attempt to define what the user specifically does upon being notified of pending urgent data, but the general notion is that the receiving process will take action to process the urgent data quickly.

2.9. Precedence and Security.

The TCP makes use of the internet protocol type of service field and security option to provide precedence and security on a per connection basis to TCP users. Not all TCP modules will necessarily function in a multilevel secure environment; some may be limited to unclassified use only, and others may operate at only one security level and compartment. Consequently, some TCP implementations and services to users may be limited to a subset of the multilevel secure case.

TCP modules which operate in a multilevel secure environment must properly mark outgoing segments with the security, compartment, and precedence. Such TCP modules must also provide to their users or higher level protocols such as Telnet or THP an interface to allow them to specify the desired security level, compartment, and precedence of connections.

2.10. Robustness Principle

TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others.

Transmission Control Protocol

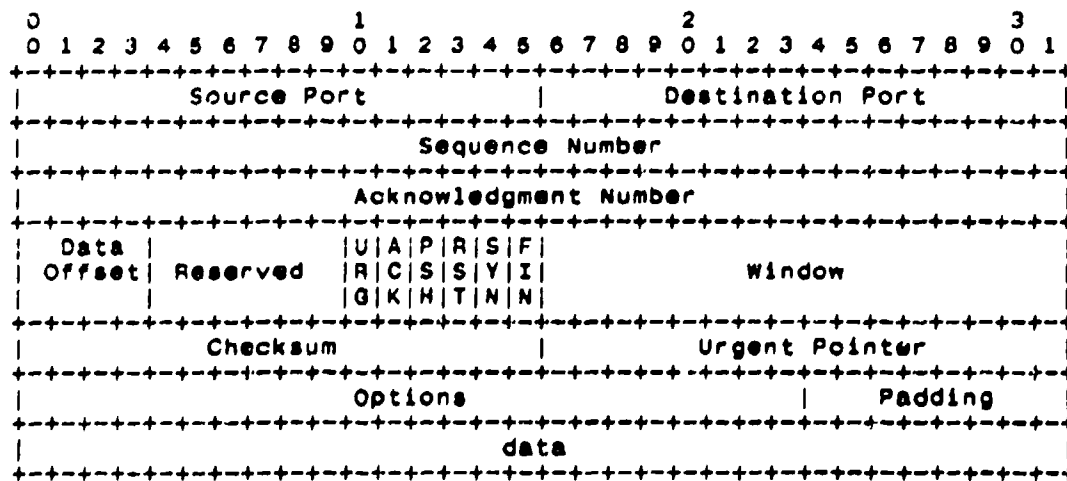
September 1981

3. FUNCTIONAL SPECIFICATION

3.1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol header carries several information fields, including the source and destination host addresses [2]. A TCP header follows the internet header, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP.

TCP Header Format



TCP Header Format

Note that one tick mark represents one bit position.

Figure 3.

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.

September 1981

Transmission Control Protocol
Functional Specification

Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits

If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Reserved: 6 bits

Reserved for future use. Must be zero.

Control Bits: 6 bits (from left to right):

URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

Window: 16 bits

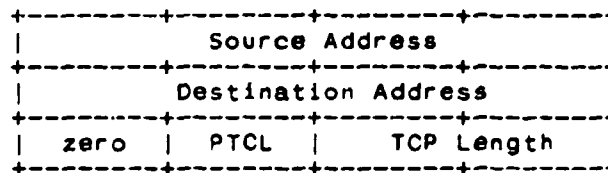
The number of data octets beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. If a segment contains an odd number of header and text octets to be checksummed, the last octet is padded on the right with zeros to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a 96 bit pseudo header conceptually

prefixed to the TCP header. This pseudo header contains the Source Address, the Destination Address, the Protocol, and TCP length. This gives the TCP protection against misrouted segments. This information is carried in the Internet Protocol and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP on the IP.



The TCP Length is the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only to be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

A TCP must implement all options.

September 1981

Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

Specific Option Definitions

End of Option List

```
+-----+
|00000000|
+-----+
Kind=0
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

```
+-----+
|00000001|
+-----+
Kind=1
```

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size

```
+-----+-----+-----+-----+
|00000010|00000100| max seg size |
+-----+-----+-----+-----+
Kind=2 Length=4
```

September 1981

Transmission Control Protocol
Functional Specification

Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.

Padding: variable

The TCP header padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

3.2. Terminology

Before we can discuss very much about the operation of the TCP we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB. Among the variables stored in the TCB are the local and remote socket numbers, the security and precedence of the connection, pointers to the user's send and receive buffers, pointers to the retransmit queue and to the current segment. In addition several variables relating to the send and receive sequence numbers are stored in the TCB.

Send Sequence Variables

SND.UNA - send unacknowledged
SND.NXT - send next
SND.WND - send window
SND.UP - send urgent pointer
SND.WL1 - segment sequence number used for last window update
SND.WL2 - segment acknowledgment number used for last window update
ISS - initial send sequence number

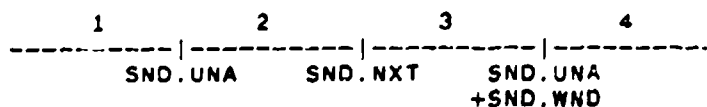
Receive Sequence Variables

RCV.NXT - receive next
RCV.WND - receive window
RCV.UP - receive urgent pointer
IRS - initial receive sequence number

September 1981

The following diagrams may help to relate some of these variables to the sequence space.

Send Sequence Space



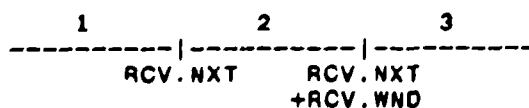
- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers of unacknowledged data
- 3 - sequence numbers allowed for new data transmission
- 4 - future sequence numbers which are not yet allowed

Send Sequence Space

Figure 4.

The send window is the portion of the sequence space labeled 3 in figure 4.

Receive Sequence Space



- 1 - old sequence numbers which have been acknowledged
- 2 - sequence numbers allowed for new reception
- 3 - future sequence numbers which are not yet allowed

Receive Sequence Space

Figure 5.

The receive window is the portion of the sequence space labeled 2 in figure 5.

There are also some variables used frequently in the discussion that take their values from the fields of the current segment.

September 1981

Transmission Control Protocol
Functional Specification

Current Segment Variables

SEG.SEQ - segment sequence number
SEG.ACK - segment acknowledgment number
SEG.LEN - segment length
SEG.WND - segment window
SEG.UP - segment urgent pointer
SEG.PRC - segment precedence value

A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection. Briefly the meanings of the states are:

LISTEN - represents waiting for a connection request from any remote TCP and port.

SYN-SENT - represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED - represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED - represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 - represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 - represents waiting for a connection termination request from the remote TCP.

CLOSE-WAIT - represents waiting for a connection termination request from the local user.

CLOSING - represents waiting for a connection termination request acknowledgment from the remote TCP.

LAST-ACK - represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).

September 1981

TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request.

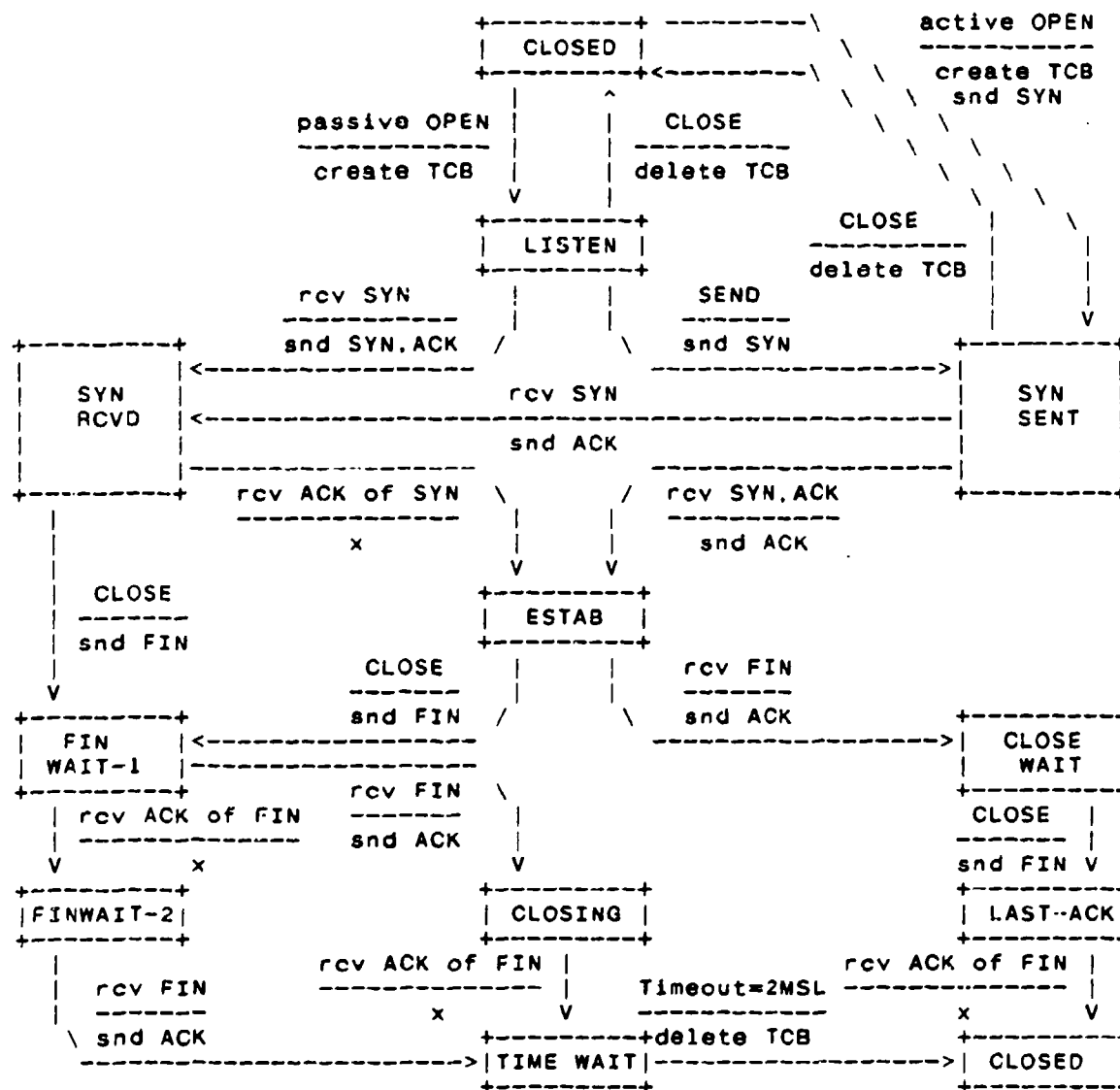
CLOSED - represents no connection state at all.

A TCP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE, ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

The state diagram in figure 6 illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions which are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP to events.

NOTE BENE: this diagram is only a summary and must not be taken as the total specification.

Transmission Control Protocol
Functional Specification



TCP Connection State Diagram
Figure 6.

3.3. Sequence Numbers

A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number. Since every octet is sequenced, each of them can be acknowledged. The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received. This mechanism allows for straight-forward duplicate detection in the presence of retransmission. Numbering of octets within a segment is that the first data octet immediately following the header is the lowest numbered, and the following octets are numbered consecutively.

It is essential to remember that the actual sequence number space is finite, though very large. This space ranges from 0 to $2^{32} - 1$. Since the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2^{32} . This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from $2^{32} - 1$ to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. The symbol " $=<$ " means "less than or equal" (modulo 2^{32}).

The typical kinds of sequence number comparisons which the TCP must perform include:

- (a) Determining that an acknowledgment refers to some sequence number sent but not yet acknowledged.
- (b) Determining that all sequence numbers occupied by a segment have been acknowledged (e.g., to remove the segment from a retransmission queue).
- (c) Determining that an incoming segment contains sequence numbers which are expected (i.e., that the segment "overlaps" the receive window).

September 1981

Transmission Control Protocol
Functional Specification

In response to sending data the TCP will receive acknowledgments. The following comparisons are needed to process the acknowledgments.

SND.UNA = oldest unacknowledged sequence number

SND.NXT = next sequence number to be sent

SEG.ACK = acknowledgment from the receiving TCP (next sequence number expected by the receiving TCP)

SEG.SEQ = first sequence number of a segment

SEG.LEN = the number of octets occupied by the data in the segment (counting SYN and FIN)

SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

A new acknowledgment (called an "acceptable ack"), is one for which the inequality below holds:

$$\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$$

A segment on the retransmission queue is fully acknowledged if the sum of its sequence number and length is less or equal than the acknowledgment value in the incoming segment.

When data is received the following comparisons are needed:

RCV.NXT = next sequence number expected on an incoming segments, and is the left or lower edge of the receive window

$\text{RCV.NXT} + \text{RCV.WND} - 1$ = last sequence number expected on an incoming segment, and is the right or upper edge of the receive window

SEG.SEQ = first sequence number occupied by the incoming segment

$\text{SEG.SEQ} + \text{SEG.LEN} - 1$ = last sequence number occupied by the incoming segment

A segment is judged to occupy a portion of valid receive sequence space if

$$\text{RCV.NXT} \leq \text{SEG.SEQ} < \text{RCV.NXT} + \text{RCV.WND}$$

or

$$\text{RCV.NXT} \leq \text{SEG.SEQ} + \text{SEG.LEN} - 1 < \text{RCV.NXT} + \text{RCV.WND}$$

September 1981

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT ≤ SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

Note that when the receive window is zero no segments should be acceptable except ACK segments. Thus, it is possible for a TCP to maintain a zero receive window while transmitting data and receiving ACKs. However, even when the receive window is zero, a TCP must process the RST and URG fields of all incoming segments.

We have taken advantage of the numbering scheme to protect certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be retransmitted and acknowledged without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The SYN and FIN are the only controls requiring this protection, and these controls are used only at connection opening and closing. For sequence number purposes, the SYN is considered to occur before the first actual data octet of the segment in which it occurs, while the FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (SEG.LEN) includes both data and sequence space occupying controls. When a SYN is present then SEG.SEQ is the sequence number of the SYN.

Initial Sequence Number Selection

The protocol places no restriction on a particular connection being used over and over again. A connection is defined by a pair of sockets. New instances of a connection will be referred to as incarnations of the connection. The problem that arises from this is -- "how does the TCP identify duplicate segments from previous incarnations of the connection?" This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished.

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN. The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4 microseconds. Thus, the ISN cycles approximately every 4.55 hours. Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number (ISS) is chosen by the data sending TCP, and the initial receive sequence number (IRS) is learned during the connection establishing procedure.

For a connection to be established or initialized, the two TCPs must synchronize on each other's initial sequence numbers. This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (for synchronize) and the initial sequence numbers. As a shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISN's.

The synchronization requires each side to send it's own initial sequence number and to receive a confirmation of it in acknowledgment from the other side. Each side must also receive the other side's initial sequence number and send a confirming acknowledgment.

- 1) A --> B SYN my sequence number is X
- 2) A <-- B ACK your sequence number is X
- 3) A <-- B SYN my sequence number is Y
- 4) A --> B ACK your sequence number is Y

September 1981

Because steps 2 and 3 can be combined in a single message this is called the three way (or three message) handshake.

A three way handshake is necessary because sequence numbers are not tied to a global clock in the network, and TCPs may have different mechanisms for picking the ISN's. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN. The three way handshake and the advantages of a clock-driven scheme are discussed in [3].

Knowing When to Keep Quiet

To be sure that a TCP does not create a segment that carries a sequence number which may be duplicated by an old segment remaining in the network, the TCP must keep quiet for a maximum segment lifetime (MSL) before assigning any sequence numbers upon starting up or recovering from a crash in which memory of sequence numbers in use was lost. For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. Note that if a TCP is reinitialized in some sense, yet retains its memory of sequence numbers in use, then it need not wait at all; it must only be sure to use sequence numbers larger than those recently used.

The TCP Quiet Time Concept

This specification provides that hosts which "crash" without retaining any knowledge of the last sequence numbers transmitted on each active (i.e., not closed) connection shall delay emitting any TCP segments for at least the agreed Maximum Segment Lifetime (MSL) in the internet system of which the host is a part. In the paragraphs below, an explanation for this specification is given. TCP implementors may violate the "quiet time" restriction, but only at the risk of causing some old data to be accepted as new or new data rejected as old duplicated by some receivers in the internet system.

TCPs consume sequence number space each time a segment is formed and entered into the network output queue at a source host. The duplicate detection and sequencing algorithm in the TCP protocol relies on the unique binding of segment data to sequence space to the extent that sequence numbers will not cycle through all 2^{32} values before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be

September 1981

Transmission Control Protocol
Functional Specification

assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. Remember that each segment is bound to as many consecutive sequence numbers as there are octets of data in the segment.

Under normal conditions, TCPs keep track of the next sequence number to emit and the oldest awaiting acknowledgment so as to avoid mistakenly using a sequence number over before its first use has been acknowledged. This alone does not guarantee that old duplicate data is drained from the net, so the sequence space has been made very large to reduce the probability that a wandering duplicate will cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours to use up 2^{32} octets of sequence space. Since the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this is deemed ample protection for foreseeable nets, even if data rates escalate to 10's of megabits/sec. At 100 megabits/sec. the cycle time is 5.4 minutes which may be a little short, but still within reason.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP were to start all connections with sequence number 0, then upon crashing and restarting, a TCP might re-form an earlier connection (possibly after half-open connection resolution) and emit packets with sequence numbers identical to or overlapping with packets still in the network which were emitted on an earlier incarnation of the same connection. In the absence of knowledge about the sequence numbers used on a particular connection, the TCP specification recommends that the source delay for MSL seconds before emitting segments on the connection, to allow time for segments from the earlier connection incarnation to drain from the system.

Even hosts which can remember the time of day and used it to select initial sequence number values are not immune from this problem (i.e., even if time of day is used to select an initial sequence number for each new connection incarnation).

Suppose, for example, that a connection is opened starting with sequence number S. Suppose that this connection is not used much and that eventually the initial sequence number function (ISN(t)) takes on a value equal to the sequence number, say S1, of the last segment sent by this TCP on a particular connection. Now suppose, at this instant, the host crashes, recovers, and establishes a new incarnation of the connection. The initial sequence number chosen is $S1 = \text{ISN}(t)$ -- last used sequence number on old incarnation of connection! If the recovery occurs quickly enough, any old

September 1981

duplicates in the net bearing sequence numbers in the neighborhood of S_1 may arrive and be treated as new packets by the receiver of the new incarnation of the connection.

The problem is that the recovering host may not know for how long it crashed nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

One way to deal with this problem is to deliberately delay emitting segments for one MSL after recovery from a crash-- this is the "quite time" specification. Hosts which prefer to avoid waiting are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quite time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a crash, or may informally implement the "quite time" for all connections. Obviously, even where a user selects to "wait," this is not necessary after the host has been "up" for at least MSL seconds.

To summarize: every segment emitted occupies one or more sequence numbers in the sequence space, the numbers occupied by a segment are "busy" or "in use" until MSL seconds have passed, upon crashing a block of space-time is occupied by the octets of the last emitted segment. If a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of the last segment of the previous connection incarnation, there is a potential sequence number overlap area which could cause confusion at the receiver.

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP doesn't deliver the data to the user until it is clear the data is valid (i.e., the data must be buffered at the receiver until the connection reaches the ESTABLISHED state). The three-way handshake reduces the possibility of false connections. It is the

September 1981

Transmission Control Protocol
Functional Specification

implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure 7 below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (→) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (←), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). An "XXX" indicates a segment which is lost or rejected. Comments appear in parentheses. TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT	→ <SEQ=100><CTL=SYN>	→ SYN-RECEIVED
3. ESTABLISHED	← <SEQ=300><ACK=101><CTL=SYN,ACK>	← SYN-RECEIVED
4. ESTABLISHED	→ <SEQ=101><ACK=301><CTL=ACK>	→ ESTABLISHED
5. ESTABLISHED	→ <SEQ=101><ACK=301><CTL=ACK><DATA>	→ ESTABLISHED

Basic 3-Way Handshake for Connection Synchronization

Figure 7.

In line 2 of figure 7, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).

September 1981

Simultaneous initiation is only slightly more complex, as is shown in figure 8. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

TCP A		TCP B
1. CLOSED		CLOSED
2. SYN-SENT	--> <SEQ=100><CTL=SYN>	...
3. SYN-RECEIVED	<-- <SEQ=300><CTL=SYN>	<-- SYN-SENT
4.	... <SEQ=100><CTL=SYN>	--> SYN-RECEIVED
5. SYN-RECEIVED	--> <SEQ=100><ACK=301><CTL=SYN,ACK>	...
6. ESTABLISHED	<-- <SEQ=300><ACK=101><CTL=SYN,ACK>	<-- SYN-RECEIVED
7.	... <SEQ=101><ACK=301><CTL=ACK>	--> ESTABLISHED

Simultaneous Connection Synchronization

Figure 8.

The principle reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, has been devised. If the receiving TCP is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user. We discuss this latter case under "half-open" connections below.

TCP A		TCP B
1. CLOSED		LISTEN
2. SYN-SENT --> <SEQ=100><CTL=SYN>		...
3. (duplicate) ... <SEQ=90><CTL=SYN>		--> SYN-RECEIVED
4. SYN-SENT <-- <SEQ=300><ACK=91><CTL=SYN,ACK>		<-- SYN-RECEIVED
5. SYN-SENT --> <SEQ=91><CTL=RST>		--> LISTEN
6. ... <SEQ=100><CTL=SYN>		--> SYN-RECEIVED
7. SYN-SENT <-- <SEQ=400><ACK=101><CTL=SYN,ACK>		<-- SYN-RECEIVED
8. ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>		--> ESTABLISHED

Recovery from Old Duplicate SYN

Figure 9.

As a simple example of recovery from old duplicates, consider figure 9. At line 3, an old duplicate SYN arrives at TCP B. TCP B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP B, on receiving the RST, returns to the LISTEN state. When the original SYN (pun intended) finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.

Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCPs has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a crash that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual, and the recovery procedure is mildly involved.

If at site A the connection no longer exists, then an attempt by the

September 1981

user at site B to send any data on it will result in the site B TCP receiving a reset control message. Such a message indicates to the site B TCP that something is wrong, and it is expected to abort the connection.

Assume that two user processes A and B are communicating with one another when a crash occurs causing loss of memory to A's TCP. Depending on the operating system supporting A's TCP, it is likely that some error recovery mechanism exists. When the TCP is up again, A is likely to start again from the beginning or from a recovery point. As a result, A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (A's) TCP. In an attempt to establish the connection, A's TCP will send a segment containing SYN. This scenario leads to the example shown in figure 10. After TCP A crashes, the user attempts to re-open the connection. TCP B, in the meantime, thinks the connection is open.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. CLOSED	ESTABLISHED
3. SYN-SENT --> <SEQ=400><CTL=SYN>	--> (??)
4. (!!) <-- <SEQ=300><ACK=100><CTL=ACK>	<-- ESTABLISHED
5. SYN-SENT --> <SEQ=100><CTL=RST>	--> (Abort!!)
6. SYN-SENT	CLOSED
7. SYN-SENT --> <SEQ=400><CTL=SYN>	-->

Half-Open Connection Discovery

Figure 10.

When the SYN arrives at line 3, TCP B, being in a synchronized state, and the incoming segment outside the window, responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP B aborts at line 5. TCP A will

continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake of figure 7.

An interesting alternative case occurs when TCP A crashes and TCP B tries to send data on what it thinks is a synchronized connection. This is illustrated in figure 11. In this case, the data arriving at TCP A from TCP B (line 2) is unacceptable because no such connection exists, so TCP A sends a RST. The RST is acceptable so TCP B processes it and aborts the connection.

TCP A	TCP B
1. (CRASH)	(send 300, receive 100)
2. (??) <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK>	<-- ESTABLISHED
3. --> <SEQ=100><CTL=RST>	--> (ABORT!!)

Active Side Causes Half-Open Connection Discovery

Figure 11.

In figure 12, we find the two TCPs A and B with passive connections waiting for SYN. An old duplicate arriving at TCP B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP B accepts the reset and returns to its passive LISTEN state.

TCP A	TCP B
1. LISTEN	LISTEN
2. ... <SEQ=Z><CTL=SYN>	--> SYN-RECEIVED
3. (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>	<-- SYN-RECEIVED
4. --> <SEQ=Z+1><CTL=RST>	--> (return to LISTEN!)
5. LISTEN	LISTEN

Old Duplicate SYN Initiates a Reset on two Passive Sockets

Figure 12.

September 1981

A variety of other cases are possible, all of which are accounted for by the following rules for RST generation and processing.

Reset Generation

As a general rule, reset (RST) must be sent whenever a segment arrives which apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

There are three groups of states:

1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. In particular, SYNs addressed to a non-existent connection are rejected by this means.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), or if an incoming segment has a security level or compartment which does not exactly match the level and compartment requested for the connection, a reset is sent.

If our SYN has not been acknowledged and the precedence level of the incoming segment is higher than the precedence level requested then either raise the local precedence level (if allowed by the user and the system) or send a reset; or if the precedence level of the incoming segment is lower than the precedence level requested then continue as if the precedence matched exactly (if the remote TCP cannot raise the precedence level to match ours this will be detected in the next segment it sends, and the connection will be terminated then). If our SYN has been acknowledged (perhaps in this incoming segment) the precedence level of the incoming segment must match the local precedence level exactly, if it does not a reset must be sent.

If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the same state.

September 1981

Transmission Control Protocol
Functional Specification

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptable acknowledgment number) must elicit only an empty acknowledgment segment containing the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

If an incoming segment has a security level, or compartment, or precedence which does not exactly match the level, and compartment, and precedence requested for the connection, a reset is sent and connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.

3.5. Closing a Connection

CLOSE is an operation meaning "I have no more data to send." The notion of closing a full-duplex connection is subject to ambiguous interpretation, of course, since it may not be obvious how to treat the receiving side of the connection. We have chosen to treat CLOSE in a simplex fashion. The user who CLOSEs may continue to RECEIVE until he is told that the other side has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that a RECEIVE failed because the other side has CLOSED. We assume that the TCP will signal a user, even if no RECEIVES are outstanding, that the other side has closed, so the user can terminate his side gracefully. A TCP will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all his data was received at the destination TCP. Users must keep reading connections they close for sending until the TCP says no more data.

September 1981

There are essentially three cases:

- 1) The user initiates by telling the TCP to CLOSE the connection
- 2) The remote TCP initiates by sending a FIN control signal
- 3) Both users CLOSE simultaneously

Case 1: Local user initiates the close

In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP, and it enters the FIN-WAIT-1 state. RECEIVES are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP has both acknowledged the FIN and sent a FIN of its own, the first TCP can ACK this FIN. Note that a TCP receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP can send a FIN to the other TCP after sending any remaining data. The TCP then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the user timeout the connection is aborted and the user is told.

Case 3: both users close simultaneously

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged. When all segments preceding the FINs have been processed and acknowledged, each TCP can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close) FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK>	--> CLOSE-WAIT
3. FIN-WAIT-2	<-- <SEQ=300><ACK=101><CTL=ACK>	<-- CLOSE-WAIT
4. TIME-WAIT	<-- <SEQ=300><ACK=101><CTL=FIN,ACK>	(Close) <-- LAST-ACK
5. TIME-WAIT	--> <SEQ=101><ACK=301><CTL=ACK>	--> CLOSED
6. (2 MSL) CLOSED		

Normal Close Sequence

Figure 13.

TCP A		TCP B
1. ESTABLISHED		ESTABLISHED
2. (Close) FIN-WAIT-1	--> <SEQ=100><ACK=300><CTL=FIN,ACK> <-- <SEQ=300><ACK=100><CTL=FIN,ACK> ... <SEQ=100><ACK=300><CTL=FIN,ACK>	(Close) ... FIN-WAIT-1 <-- -->
3. CLOSING	--> <SEQ=101><ACK=301><CTL=ACK> <-- <SEQ=301><ACK=101><CTL=ACK> ... <SEQ=101><ACK=301><CTL=ACK>	... CLOSING <-- -->
4. TIME-WAIT (2 MSL) CLOSED		TIME-WAIT (2 MSL) CLOSED

Simultaneous Close Sequence

Figure 14.

September 1981

3.6. Precedence and Security

The intent is that connection be allowed only between ports operating with exactly the same security and compartment values and at the higher of the precedence level requested by the two ports.

The precedence and security parameters used in TCP are exactly those defined in the Internet Protocol (IP) [2]. Throughout this TCP specification the term "security/compartment" is intended to indicate the security parameters used in IP including security, compartment, user group, and handling restriction.

A connection attempt with mismatched security/compartment values or a lower precedence value must be rejected by sending a reset. Rejecting a connection due to too low a precedence only occurs after an acknowledgment of the SYN has been received.

Note that TCP modules which operate only at the default value of precedence will still have to check the precedence of incoming segments and possibly raise the precedence level they use on the connection.

The security parameters may be used even in a non-secure environment (the values would indicate unclassified data), thus hosts in non-secure environments must be prepared to receive the security parameters, though they need not send them.

3.7. Data Communication

Once the connection is established data is communicated by the exchange of segments. Because segments may be lost due to errors (checksum test failure), or network congestion, TCP uses retransmission (after a timeout) to ensure delivery of every segment. Duplicate segments may arrive due to network or TCP retransmission. As discussed in the section on sequence numbers the TCP performs certain tests on the sequence and acknowledgment numbers in the segments to verify their acceptability.

The sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent has been acknowledged then the three variables will be equal.

When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an

September 1981

Transmission Control Protocol Functional Specification

acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication. The amount by which the variables are advanced is the length of the data in the segment. Note that once in the ESTABLISHED state all segments must carry current acknowledgment information.

The CLOSE user call implies a push function, as does the FIN control flag in an incoming segment.

Retransmission Timeout

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout must be dynamically determined. One procedure for determining a retransmission time out is given here as an illustration.

An Example Retransmission Timeout Procedure

Measure the elapsed time between sending a data octet with a particular sequence number and receiving an acknowledgment that covers that sequence number (segments sent do not have to match segments received). This measured elapsed time is the Round Trip Time (RTT). Next compute a Smoothed Round Trip Time (SRTT) as:

$$SRTT = (\text{ALPHA} * SRTT) + ((1-\text{ALPHA}) * RTT)$$

and based on this, compute the retransmission timeout (RTO) as:

$$RTO = \min[\text{UBOUND}, \max[\text{LBOUND}, (\text{BETA} * SRTT)]]$$

where UBOUND is an upper bound on the timeout (e.g., 1 minute), LBOUND is a lower bound on the timeout (e.g., 1 second), ALPHA is a smoothing factor (e.g., .8 to .9), and BETA is a delay variance factor (e.g., 1.3 to 2.0).

The Communication of Urgent Information

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data and to permit the receiving TCP to indicate to the receiving user when all the currently known urgent data has been received by the user.

This mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the receive sequence number (RCV.NXT) at the receiving TCP, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP must tell user to go

September 1981

into "normal mode". If the urgent pointer is updated while the user is in "urgent mode", the update will be invisible to the user.

The method employs a urgent field which is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful and must be added to the segment sequence number to yield the urgent pointer. The absence of this flag indicates that there is no urgent data outstanding.

To send an urgent indication the user must also send at least one data octet. If the sending user also indicates a push, timely delivery of the urgent information to the destination process is enhanced.

Managing the Window

The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCPs. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The mechanisms provided allow a TCP to advertise a large window and to subsequently advertise a much smaller window without having accepted that much data. This, so called "shrinking the window," is strongly discouraged. The robustness principle dictates that TCPs will not shrink the window themselves, but will be prepared for such behavior on the part of other TCPs.

The sending TCP must be prepared to accept from the user and send at least one octet of new data even if the send window is zero. The sending TCP must regularly retransmit to the receiving TCP even when the window is zero. Two minutes is recommended for the retransmission interval when the window is zero. This retransmission is essential to guarantee that when either TCP has a zero window the re-opening of the window will be reliably reported to the other.

When the receiving TCP has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

The sending TCP packages the data to be transmitted into segments

September 1981

Transmission Control Protocol
Functional Specification

which fit the current window, and may repackage segments on the retransmission queue. Such repackaging is not required, but may be helpful.

In a connection with a one-way data flow, the window information will be carried in acknowledgment segments that all have the same sequence number so there will be no way to reorder them if they arrive out of order. This is not a serious problem, but it will allow the window information to be on occasion temporarily based on old reports from the data receiver. A refinement to avoid this problem is to act on the window information from segments that carry the highest acknowledgment number (that is segments with acknowledgment number equal or greater than the highest previously received).

The window management procedure has significant influence on the communication performance. The following comments are suggestions to implementers.

Window Management Suggestions

Allocating a very small window causes data to be transmitted in many small segments when better performance is achieved using fewer large segments.

One suggestion for avoiding small windows is for the receiver to defer updating a window until the additional allocation is at least X percent of the maximum allocation possible for the connection (where X might be 20 to 40).

Another suggestion is for the sender to avoid sending small segments by waiting until the window is large enough before sending data. If the user signals a push function then the data must be sent even if it is a small segment.

Note that the acknowledgments should not be delayed or unnecessary retransmissions will result. One strategy would be to send an acknowledgment when a small segment arrives (with out updating the window information), and then to send another acknowledgment with new window information when the window is larger.

The segment sent to probe a zero window may also begin a break up of transmitted data into smaller and smaller segments. If a segment containing a single data octet sent to probe a zero window is accepted, it consumes one octet of the window now available. If the sending TCP simply sends as much as it can whenever the window is non zero, the transmitted data will be broken into alternating big and small segments. As time goes on, occasional pauses in the receiver making window allocation available will

September 1981

Transmission Control Protocol Functional Specification

result in breaking the big segments into a small and not quite so big pair. And after a while the data transmission will be in mostly small segments.

The suggestion here is that the TCP implementations need to actively attempt to combine small window allocations into larger windows, since the mechanisms for managing the window tend to lead to many small windows in the simplest minded implementations.

3.8. Interfaces

There are of course two interfaces of concern: the user/TCP interface and the TCP/lower-level interface. We have a fairly elaborate model of the user/TCP interface, but the interface to the lower level protocol module is left unspecified here, since it will be specified in detail by the specification of the lower level protocol. For the case that the lower level is IP we note some of the parameter values that TCPs might use.

User/TCP Interface

The following functional description of user commands to the TCP is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different TCP implementations may have different user interfaces. However, all TCPs must provide a certain minimum set of services to guarantee that all TCP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all TCP implementations.

TCP User Commands

The following sections functionally characterize a USER/TCP interface. The notation used is similar to most procedure or function calls in high level languages, but this usage is not meant to rule out trap type service calls (e.g., SVCs, UUOs, EMTs).

The user commands described below specify the basic functions the TCP must perform to support interprocess communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls. In particular, some implementations may wish to automatically OPEN a connection on the first SEND or RECEIVE issued by the user for a given connection.

September 1981

Transmission Control Protocol
Functional Specification

In providing interprocess communication facilities, the TCP must not only accept commands, but must also return information to the processes it serves. The latter consists of:

- (a) general information about a connection (e.g., interrupts, remote close, binding of unspecified foreign socket).
- (b) replies to specific user commands indicating success or various types of failure.

Open

Format: OPEN (local port, foreign socket, active/passive
[, timeout] [, precedence] [, security/compartments] [, options])
-> local connection name

We assume that the local TCP is aware of the identity of the processes it serves and will check the authority of the process to use the connection specified. Depending upon the implementation of the TCP, the local network and TCP identifiers for the source address will either be supplied by the TCP or the lower level protocol (e.g., IP). These considerations are the result of concern about security, to the extent that no TCP be able to masquerade as another one, and so on. Similarly, no process can masquerade as another without the collusion of the TCP.

If the active/passive flag is set to passive, then this is a call to LISTEN for an incoming connection. A passive open may have either a fully specified foreign socket to wait for a particular connection or an unspecified foreign socket to wait for any call. A fully specified passive call can be made active by the subsequent execution of a SEND.

A transmission control block (TCB) is created and partially filled in with data from the OPEN command parameters.

On an active OPEN command, the TCP will begin the procedure to synchronize (i.e., establish) the connection at once.

The timeout, if present, permits the caller to set up a timeout for all data submitted to TCP. If data is not successfully delivered to the destination within the timeout period, the TCP will abort the connection. The present global default is five minutes.

The TCP or some component of the operating system will verify the users authority to open a connection with the specified

September 1981

precedence or security/compartiment. The absence of precedence or security/compartiment specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartiment information is exactly the same and only if the precedence is equal to or higher than the precedence requested in the OPEN call.

The precedence for the connection is the higher of the values requested in the OPEN call and received from the incoming request, and fixed at that value for the life of the connection. Implementers may want to give the user control of this precedence negotiation. For example, the user might be allowed to specify that the precedence must be exactly matched, or that any attempt to raise the precedence be confirmed by the user.

A local connection name will be returned to the user by the TCP. The local connection name can then be used as a short hand term for the connection defined by the <local socket, foreign socket> pair.

Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag, URGENT flag [,timeout])

This call causes the data contained in the indicated user buffer to be sent on the indicated connection. If the connection has not been opened, the SEND is considered an error. Some implementations may allow users to SEND first; in which case, an automatic OPEN would be done. If the calling process is not authorized to use this connection, an error is returned.

If the PUSH flag is set, the data must be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer. If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency.

If the URGENT flag is set, segments sent to the destination TCP will have the urgent pointer set. The receiving TCP will signal the urgent condition to the receiving process if the urgent pointer indicates that data preceding the urgent pointer has not been consumed by the receiving process. The purpose of urgent is to stimulate the receiver to process the urgent data and to indicate to the receiver when all the currently known urgent

September 1981

Transmission Control Protocol
Functional Specification

data has been received. The number of times the sending user's TCP signals urgent will not necessarily be equal to the number of times the receiving user will be notified of the presence of urgent data.

If no foreign socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a foreign segment arriving for the local socket), then the designated buffer is sent to the implied foreign socket. Users who make use of OPEN with an unspecified foreign socket can make use of SEND without ever explicitly knowing the foreign socket address.

However, if a SEND is attempted before the foreign socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. In some implementations the TCP may notify the user when an unspecified socket is bound.

If a timeout is specified, the current user timeout for this connection is changed to the new one.

In the simplest implementation, SEND would not return control to the sending process until either the transmission was complete or the timeout had been exceeded. However, this simple method is both subject to deadlocks (for example, both sides of the connection might try to do SENDs before doing any RECEIVES) and offers poor performance, so it is not recommended. A more sophisticated implementation would return immediately to allow the process to run concurrently with network I/O, and, furthermore, to allow multiple SENDs to be in progress. Multiple SENDs are served in first come, first served order, so the TCP will queue those it cannot service immediately.

We have implicitly assumed an asynchronous user interface in which a SEND later elicits some kind of SIGNAL or pseudo-interrupt from the serving TCP. An alternative is to return a response immediately. For instance, SENDs might return immediate local acknowledgment, even if the segment sent had not been acknowledged by the distant TCP. We could optimistically assume eventual success. If we are wrong, the connection will close anyway due to the timeout. In implementations of this kind (synchronous), there will still be some asynchronous signals, but these will deal with the connection itself, and not with specific segments or buffers.

In order for the process to distinguish among error or success indications for different SENDs, it might be appropriate for the

September 1981

buffer address to be returned along with the coded response to the SEND request. TCP-to-user signals are discussed below, indicating the information which should be returned to the calling process.

Receive

Format: RECEIVE (local connection name, buffer address, byte count) -> byte count, urgent flag, push flag

This command allocates a receiving buffer associated with the specified connection. If no OPEN precedes this command or the calling process is not authorized to use this connection, an error is returned.

In the simplest implementation, control would not return to the calling program until either the buffer was filled, or some error occurred, but this scheme is highly subject to deadlocks. A more sophisticated implementation would permit several RECEIVES to be outstanding at once. These would be filled as segments arrive. This strategy permits increased throughput at the cost of a more elaborate scheme (possibly asynchronous) to notify the calling program that a PUSH has been seen or a buffer filled.

If enough data arrive to fill the buffer before a PUSH is seen, the PUSH flag will not be set in the response to the RECEIVE. The buffer will be filled with as much data as it can hold. If a PUSH is seen before the buffer is filled the buffer will be returned partially filled and PUSH indicated.

If there is urgent data the user will have been informed as soon as it arrived via a TCP-to-user signal. The receiving user should thus be in "urgent mode". If the URGENT flag is on, additional urgent data remains. If the URGENT flag is off, this call to RECEIVE has returned all the urgent data, and the user may now leave "urgent mode". Note that data following the urgent pointer (non-urgent data) cannot be delivered to the user in the same buffer with preceding urgent data unless the boundary is clearly marked for the user.

To distinguish among several outstanding RECEIVES and to take care of the case that a buffer is not completely filled, the return code is accompanied by both a buffer pointer and a byte count indicating the actual length of the data received.

Alternative implementations of RECEIVE might have the TCP

September 1981

Transmission Control Protocol
Functional Specification

allocate buffer storage, or the TCP might share a ring buffer with the user.

Close

Format: CLOSE (local connection name)

This command causes the connection specified to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Closing connections is intended to be a graceful operation in the sense that outstanding SENDs will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several SEND calls, followed by a CLOSE, and expect all the data to be sent to the destination. It should also be clear that users should continue to RECEIVE on CLOSING connections, since the other side may be trying to transmit the last of its data. Thus, CLOSE means "I have no more to send" but does not mean "I will not receive any more." It may happen (if the user level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, CLOSE turns into ABORT, and the closing TCP gives up.

The user may CLOSE the connection at any time on his own initiative, or in response to various prompts from the TCP (e.g., remote close executed, transmission timeout exceeded, destination inaccessible).

Because closing a connection requires communication with the foreign TCP, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP replies to the CLOSE command will result in error responses.

Close also implies push function.

Status

Format: STATUS (local connection name) -> status data

This is an implementation dependent user command and could be excluded without adverse effect. Information returned would typically come from the TCB associated with the connection.

This command returns a data block containing the following information:

local socket.

September 1981

foreign socket,
local connection name,
receive window,
send window,
connection state,
number of buffers awaiting acknowledgment,
number of buffers pending receipt,
urgent state,
precedence,
security/compartments,
and transmission timeout.

Depending on the state of the connection, or on the implementation itself, some of this information may not be available or meaningful. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

Abort

Format: ABORT (local connection name)

This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the TCP on the other side of the connection. Depending on the implementation, users may receive abort indications for each outstanding SEND or RECEIVE, or may simply receive an ABORT-acknowledgment.

TCP-to-User Messages

It is assumed that the operating system environment provides a means for the TCP to asynchronously signal the user program. When the TCP does signal a user program, certain information is passed to the user. Often in the specification the information will be an error message. In other cases there will be information relating to the completion of processing a SEND or RECEIVE or other user call.

The following information is provided:

Local Connection Name	Always
Response String	Always
Buffer Address	Send & Receive
Byte count (counts bytes received)	Receive
Push flag	Receive
Urgent flag	Receive

September 1981

Transmission Control Protocol
Functional Specification

TCP/Lower-Level Interface

The TCP calls on a lower level protocol module to actually send and receive information over a network. One case is that of the ARPA internetwork system where the lower level module is the Internet Protocol (IP) [2].

If the lower level protocol is IP it provides arguments for a type of service and for a time to live. TCP uses the following settings for these parameters:

Type of Service = Precedence: routine, Delay: normal, Throughput: normal, Reliability: normal; or 00000000.

Time to Live = one minute, or 00111100.

Note that the assumed maximum segment lifetime is two minutes. Here we explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute.

If the lower level is IP (or other protocol that provides this feature) and source routing is used, the interface must allow the route information to be communicated. This is especially important so that the source and destination addresses used in the TCP checksum be the originating source and ultimate destination. It is also important to preserve the return route to answer connection requests.

Any lower level protocol will have to provide the source address, destination address, and protocol fields, and some way to determine the "TCP length", both to provide the functional equivalent service of IP and to be used in the TCP checksum.

3.9. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

User Calls

- OPEN
- SEND
- RECEIVE
- CLOSE
- ABORT
- STATUS

Arriving Segments

- SEGMENT ARRIVES

Timeouts

- USER TIMEOUT
- RETRANSMISSION TIMEOUT
- TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses are given as character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, et cetera, is modulo 2^{32} the size of the sequence number space. Also note that " $=<$ " means less than or equal to (modulo 2^{32}).

September 1981

Transmission Control Protocol
Functional Specification

A natural way to think about processing incoming segments is to imagine that they are first tested for proper sequence number (i.e., that their contents lie in the range of the expected "receive window" in the sequence number space) and then that they are generally queued and processed in sequence number order.

When a segment overlaps other already received segments we reconstruct the segment to contain just the new data, and adjust the header fields to be consistent.

Note that if no state change is mentioned the TCP stays in the same state.

September 1981

OPEN Call

OPEN Call

CLOSED STATE (i.e., TCB does not exist)

Create a new transmission control block (TCB) to hold connection state information. Fill in local socket identifier, foreign socket, precedence, security/compartments, and user timeout information. Note that some parts of the foreign socket may be unspecified in a passive OPEN and are to be filled in by the parameters of the incoming SYN segment. Verify the security and precedence requested are allowed for this user, if not return "error: precedence not allowed" or "error: security/compartments not allowed." If passive enter the LISTEN state and return. If active and the foreign socket is unspecified, return "error: foreign socket unspecified"; if active and the foreign socket is specified, issue a SYN segment. An initial send sequence number (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN> is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT state, and return.

If the caller does not have access to the local socket specified, return "error: connection illegal for this process". If there is no room to create a new connection, return "error: insufficient resources".

LISTEN STATE

If active and the foreign socket is specified, change the connection from passive to active, select a local socket. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If foreign socket was not specified, then return "error: foreign socket unspecified".

September 1981

Transmission Control Protocol
Functional Specification

OPEN Call

SYN-SENT STATE
SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection already exists".

September 1981

SEND Call

SEND Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, then return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

If the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: foreign socket unspecified".

SYN-SENT STATE

SYN-RECEIVED STATE

Queue the data for transmission after entering ESTABLISHED state. If no space to queue, respond with "error: insufficient resources".

ESTABLISHED STATE

CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). If there is insufficient space to remember this buffer, simply return "error: insufficient resources".

If the urgent flag is set, then $SND.UP \leftarrow SND.NXT - 1$ and set the urgent pointer in the outgoing segments.

September 1981

Transmission Control Protocol
Functional Specification

SEND Call

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing" and do not service request.

September 1981

RECEIVE Call

RECEIVE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return
"error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE
SYN-SENT STATE
SYN-RECEIVED STATE

Queue for processing after entering ESTABLISHED state. If there
is no room to queue this request, respond with "error:
insufficient resources".

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If insufficient incoming segments are queued to satisfy the
request, queue the request. If there is no queue space to
remember the RECEIVE, respond with "error: insufficient
resources".

Reassemble queued incoming segments into receive buffer and return
to user. Mark "push seen" (PUSH) if this is the case.

If RCV.UP is in advance of the data currently being passed to the
user notify the user of the presence of urgent data.

When the TCP takes responsibility for delivering data to the user
that fact must be communicated to the sender via an
acknowledgment. The formation of such an acknowledgment is
described below in the discussion of processing an incoming
segment.

September 1981

Transmission Control Protocol
Functional Specification

RECEIVE Call

CLOSE-WAIT STATE

Since the remote side has already sent FIN, RECEIVES must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing".

September 1981

CLOSE Call

CLOSE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES are returned with "error: closing" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

Delete the TCB and return "error: closing" responses to any queued SENDs, or RECEIVES.

SYN-RECEIVED STATE

If no SENDs have been issued and there is no pending data to send, then form a FIN segment and send it, and enter FIN-WAIT-1 state; otherwise queue for processing after entering ESTABLISHED state.

ESTABLISHED STATE

Queue this until all preceding SENDs have been segmentized, then form a FIN segment and send it. In any case, enter FIN-WAIT-1 state.

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

Strictly speaking, this is an error and should receive a "error: connection closing" response. An "ok" response would be acceptable, too, as long as a second FIN is not emitted (the first FIN may be retransmitted though).

September 1981

Transmission Control Protocol
Functional Specification

CLOSE Call

CLOSE-WAIT STATE

Queue this request until all preceding SENDs have been
segmentized; then send a FIN segment, enter CLOSING state.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT STATE

Respond with "error: connection closing".

September 1981

ABORT Call

ABORT Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVES should be returned with "error: connection reset" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

All queued SENDs and RECEIVES should be given "connection reset" notification, delete the TCB, enter CLOSED state, and return.

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE

Send a reset segment:

<SEQ=SNQ.NXT><CTL=RST>

All queued SENDs and RECEIVES should be given "connection reset" notification; all segments queued for transmission (except for the RST formed above) or retransmission should be flushed, delete the TCB, enter CLOSED state, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Respond with "ok" and delete the TCB, enter CLOSED state, and return.

September 1981

Transmission Control Protocol
Functional Specification

STATUS Call

STATUS Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return
"error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Return "state = LISTEN", and the TCB pointer.

SYN-SENT STATE

Return "state = SYN-SENT", and the TCB pointer.

SYN-RECEIVED STATE

Return "state = SYN-RECEIVED", and the TCB pointer.

ESTABLISHED STATE

Return "state = ESTABLISHED", and the TCB pointer.

FIN-WAIT-1 STATE

Return "state = FIN-WAIT-1", and the TCB pointer.

FIN-WAIT-2 STATE

Return "state = FIN-WAIT-2", and the TCB pointer.

CLOSE-WAIT STATE

Return "state = CLOSE-WAIT", and the TCB pointer.

CLOSING STATE

Return "state = CLOSING", and the TCB pointer.

LAST-ACK STATE

Return "state = LAST-ACK", and the TCB pointer.

Transmission Control Protocol
Functional Specification

September 1981

STATUS Call

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP that sent the offending segment.

If the ACK bit is off, sequence number zero is used.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the ACK bit is on,

<SEQ=SEG.ACK><CTL=RST>

Return.

If the state is LISTEN then

first check for an RST

An incoming RST should be ignored. Return.

second check for an ACK

Any acknowledgment is bad if it arrives on a connection still in the LISTEN state. An acceptable reset segment should be formed for any arriving ACK-bearing segment. The RST should be formatted as follows:

<SEQ=SEG.ACK><CTL=RST>

Return.

third check for a SYN

If the SYN bit is set, check the security. If the security/compartments on the incoming segment does not exactly match the security/compartments in the TCB then send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

September 1981

SEGMENT ARRIVES

If the SEG.PRC is greater than the TCB.PRC then if allowed by the user and the system set TCB.PRC ← SEG.PRC, if not allowed send a reset and return.

<SEQ=SEG.ACK><CTL=RST>

If the SEG.PRC is less than the TCB.PRC then continue.

Set RCV.NXT to SEG.SEQ+1, ISS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a SYN segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the foreign socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

Any other control or text-bearing segment (not containing SYN) must have an ACK and thus would be discarded by the ACK processing. An incoming RST segment could not be valid, since it could not have been sent in response to anything sent by this incarnation of the connection. So you are unlikely to get here, but if you do, drop the segment, and return.

If the state is SYN-SENT then

first check the ACK bit

If the ACK bit is set

If SEG.ACK ≤ ISS, or SEG.ACK > SND.NXT, send a reset (unless the RST bit is set, if so drop the segment and return)

<SEQ=SEG.ACK><CTL=RST>

and discard the segment. Return.

If SND.UNA ≤ SEG.ACK ≤ SND.NXT then the ACK is acceptable.

second check the RST bit

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

If the RST bit is set

If the ACK was acceptable then signal the user "error: connection reset", drop the segment, enter CLOSED state, delete TCB, and return. Otherwise (no ACK) drop the segment and return.

third check the security and precedence

If the security/compartments in the segment does not exactly match the security/compartments in the TCB, send a reset

If there is an ACK

<SEQ=SEG.ACK><CTL=RST>

Otherwise

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If there is an ACK

The precedence in the segment must match the precedence in the TCB, if not, send a reset

<SEQ=SEG.ACK><CTL=RST>

If there is no ACK

If the precedence in the segment is higher than the precedence in the TCB then if allowed by the user and the system raise the precedence in the TCB to that in the segment, if not allowed to raise the prec then send a reset.

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If the precedence in the segment is lower than the precedence in the TCB continue.

If a reset was sent, discard the segment and return.

fourth check the SYN bit

This step should be reached only if the ACK is ok, or there is no ACK, and if the segment did not contain a RST.

If the SYN bit is on and the security/compartments and precedence

September 1981

SEGMENT ARRIVES

are acceptable then, RCV.NXT is set to SEG.SEQ+1. IRS is set to SEG.SEQ. SND.UNA should be advanced to equal SEG.ACK (if there is an ACK), and any segments on the retransmission queue which are thereby acknowledged should be removed.

If SND.UNA > ISS (our SYN has been ACKed), change the connection state to ESTABLISHED, form an ACK segment

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. Data or controls which were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a SYN,ACK segment

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

and send it. If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

Otherwise,

first check sequence number

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT ≤ SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT ≤ SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

After sending the acknowledgment, drop the unacceptable segment and return.

September 1981

SEGMENT ARRIVES

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher beginning sequence numbers may be held for later processing.

second check the RST bit.

SYN-RECEIVED STATE

If the RST bit is set

If this connection was initiated with a passive OPEN (i.e., came from the LISTEN state), then return this connection to LISTEN state and return. The user need not be informed. If this connection was initiated with an active OPEN (i.e., came from SYN-SENT state) then the connection was refused, signal the user "connection refused". In either case, all segments on the retransmission queue should be removed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

ESTABLISHED

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

If the RST bit is set then, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

CLOSING STATE

LAST-ACK STATE

TIME-WAIT

If the RST bit is set then, enter the CLOSED state, delete the TCB, and return.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

third check security and precedence

SYN-RECEIVED

If the security/compartiment and precedence in the segment do not exactly match the security/compartiment and precedence in the TCB then send a reset, and return.

ESTABLISHED STATE

If the security/compartiment and precedence in the segment do not exactly match the security/compartiment and precedence in the TCB then send a reset, any outstanding RECEIVES and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

Note this check is placed following the sequence check to prevent a segment from an old connection between these ports with a different security or precedence from causing an abort of the current connection.

fourth, check the SYN bit.

SYN-RECEIVED
ESTABLISHED STATE
FIN-WAIT STATE-1
FIN-WAIT STATE-2
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

If the SYN is in the window it is an error, send a reset, any outstanding RECEIVES and SEND should receive "reset" responses, all segment queues should be flushed, the user should also receive an unsolicited general "connection reset" signal, enter the CLOSED state, delete the TCB, and return.

If the SYN is not in the window this step would not be reached and an ack would have been sent in the first step (sequence number check).

September 1981

SEGMENT ARRIVES

fifth check the ACK field.

if the ACK bit is off drop the segment and return

if the ACK bit is on

SYN-RECEIVED STATE

If $\text{SND.UNA} \leq \text{SEG.ACK} \leq \text{SND.NXT}$ then enter ESTABLISHED state and continue processing.

If the segment acknowledgment is not acceptable, form a reset segment,

$\langle \text{SEQ} = \text{SEG.ACK} \rangle \langle \text{CTL} = \text{RST} \rangle$

and send it.

ESTABLISHED STATE

If $\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$ then, set $\text{SND.UNA} \leftarrow \text{SEG.ACK}$. Any segments on the retransmission queue which are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers which have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate ($\text{SEG.ACK} < \text{SND.UNA}$), it can be ignored. If the ACK acks something not yet sent ($\text{SEG.ACK} > \text{SND.NXT}$) then send an ACK, drop the segment, and return.

If $\text{SND.UNA} < \text{SEG.ACK} \leq \text{SND.NXT}$, the send window should be updated. If ($\text{SND.WL1} < \text{SEG.SEQ}$ or ($\text{SND.WL1} = \text{SEG.SEQ}$ and $\text{SND.WL2} \leq \text{SEG.ACK}$)), set $\text{SND.WND} \leftarrow \text{SEG.WND}$, set $\text{SND.WL1} \leftarrow \text{SEG.SEQ}$, and set $\text{SND.WL2} \leftarrow \text{SEG.ACK}$.

Note that SND.WND is an offset from SND.UNA , that SND.WL1 records the sequence number of the last segment used to update SND.WND , and that SND.WL2 records the acknowledgment number of the last segment used to update SND.WND . The check here prevents using old segments to update the window.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

FIN-WAIT-1 STATE

In addition to the processing for the ESTABLISHED state, if our FIN is now acknowledged then enter FIN-WAIT-2 and continue processing in that state.

FIN-WAIT-2 STATE

In addition to the processing for the ESTABLISHED state, if the retransmission queue is empty, the user's CLOSE can be acknowledged ("ok") but do not delete the TCB.

CLOSE-WAIT STATE

Do the same processing as for the ESTABLISHED state.

CLOSING STATE

In addition to the processing for the ESTABLISHED state, if the ACK acknowledges our FIN then enter the TIME-WAIT state, otherwise ignore the segment.

LAST-ACK STATE

The only thing that can arrive in this state is an acknowledgment of our FIN. If our FIN is now acknowledged, delete the TCB, enter the CLOSED state, and return.

TIME-WAIT STATE

The only thing that can arrive in this state is a retransmission of the remote FIN. Acknowledge it, and restart the 2 MSL timeout.

sixth, check the URG bit,

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

If the URG bit is set, $RCV.UP \leftarrow \max(RCV.UP, SEG.UP)$, and signal the user that the remote side has urgent data if the urgent pointer (RCV.UP) is in advance of the data consumed. If the user has already been signaled (or is still in the "urgent mode") for this continuous sequence of urgent data, do not signal the user again.

September 1981

SEGMENT ARRIVES

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT

This should not occur, since a FIN has been received from the remote side. Ignore the URG.

seventh, process the segment text.

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Once in the ESTABLISHED state, it is possible to deliver segment text to user RECEIVE buffers. Text from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries an PUSH flag, then the user is informed, when the buffer is returned, that a PUSH has been received.

When the TCP takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.

Once the TCP takes responsibility for the data it advances RCV.NXT over the data accepted, and adjusts RCV.WND as appropriate to the current buffer availability. The total of RCV.NXT and RCV.WND should not be reduced.

Please note the window management suggestions in section 3.7.

Send an acknowledgment of the form:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

This acknowledgment should be piggybacked on a segment being transmitted if possible, without incurring undue delay.

September 1981

Transmission Control Protocol
Functional Specification

SEGMENT ARRIVES

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

This should not occur, since a FIN has been received from the remote side. Ignore the segment text.

eighth, check the FIN bit.

Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT since the SEG.SEQ cannot be validated; drop the segment and return.

If the FIN bit is set, signal the user "connection closing" and return any pending RECEIVES with same message, advance RCV.NXT over the FIN, and send an acknowledgment for the FIN. Note that FIN implies PUSH for any segment text not yet delivered to the user.

SYN-RECEIVED STATE
ESTABLISHED STATE

Enter the CLOSE-WAIT state.

FIN-WAIT-1 STATE

If our FIN has been ACKed (perhaps in this segment), then enter TIME-WAIT, start the time-wait timer, turn off the other timers; otherwise enter the CLOSING state.

FIN-WAIT-2 STATE

Enter the TIME-WAIT state. Start the time-wait timer, turn off the other timers.

CLOSE-WAIT STATE

Remain in the CLOSE-WAIT state.

CLOSING STATE

Remain in the CLOSING state.

LAST-ACK STATE

Remain in the LAST-ACK state.

September 1981

SEGMENT ARRIVES

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait
timeout.
and return.

September 1981

Transmission Control Protocol
Functional Specification

USER TIMEOUT

USER TIMEOUT

For any state if the user timeout expires, flush all queues, signal the user "error: connection aborted due to user timeout" in general and for any outstanding calls, delete the TCB, enter the CLOSED state and return.

RETRANSMISSION TIMEOUT

For any state if the retransmission timeout expires on a segment in the retransmission queue, send the segment at the front of the retransmission queue again, reinitialize the retransmission timer, and return.

TIME-WAIT TIMEOUT

If the time-wait timeout expires on a connection delete the TCB, enter the CLOSED state and return.

GLOSSARY

1822

BBN Report 1822. "The Specification of the Interconnection of a Host and an IMP". The specification of interface between a host and the ARPANET.

ACK

A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

ARPANET message

The unit of transmission between a host and an IMP in the ARPANET. The maximum size is about 1012 octets (8096 bits).

ARPANET packet

A unit of transmission used internally in the ARPANET between IMPs. The maximum size is about 126 octets (1008 bits).

connection

A logical communication path identified by a pair of sockets.

datagram

A message sent in a packet switched computer communications network.

Destination Address

The destination address, usually the network and host identifiers.

FIN

A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

fragment

A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.

FTP

A file transfer protocol.

September 1981

Transmission Control Protocol
Glossary

header
Control information at the beginning of a message, segment, fragment, packet or block of data.

host
A computer. In particular a source or destination of messages from the point of view of the communication network.

Identification
An Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.

IMP
The Interface Message Processor, the packet switch of the ARPANET.

internet address
A source or destination address specific to the host level.

internet datagram
The unit of data exchanged between an internet module and the higher level protocol together with the internet header.

internet fragment
A portion of the data of an internet datagram with an internet header.

IP
Internet Protocol.

IRS
The Initial Receive Sequence number. The first sequence number used by the sender on a connection.

ISN
The Initial Sequence Number. The first sequence number used on a connection, (either ISS or IRS). Selected on a clock based procedure.

ISS
The Initial Send Sequence number. The first sequence number used by the sender on a connection.

leader
Control information at the beginning of a message or block of data. In particular, in the ARPANET, the control information on an ARPANET message at the host-IMP interface.

September 1981

Transmission Control Protocol
Glossary

left sequence

This is the next sequence number to be acknowledged by the data receiving TCP (or the lowest currently unacknowledged sequence number) and is sometimes referred to as the left edge of the send window.

local packet

The unit of transmission within a local network.

module

An implementation, usually in software, of a protocol or other procedure.

MSL

Maximum Segment Lifetime, the time a TCP segment can exist in the internetwork system. Arbitrarily defined to be 2 minutes.

octet

An eight bit byte.

Options

An Option field may contain several options, and each option may be several octets in length. The options are used primarily in testing situations; for example, to carry timestamps. Both the Internet Protocol and TCP provide for options fields.

packet

A package of data with a header which may or may not be logically complete. More often a physical packaging than a logical packaging of data.

port

The portion of a socket that specifies which logical input or output channel of a process is associated with the data.

process

A program in execution. A source or destination of data from the point of view of the TCP or other host-to-host protocol.

PUSH

A control bit occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving user.

RCV.NXT

receive next sequence number

RCV.UP
receive urgent pointer

RCV.WND
receive window

receive next sequence number
This is the next sequence number the local TCP is expecting to receive.

receive window
This represents the sequence numbers the local (receiving) TCP is willing to receive. Thus, the local TCP considers that segments overlapping the range RCV.NXT to $RCV.NXT + RCV.WND - 1$ carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST
A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver can determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.

RTP
Real Time Protocol: A host-to-host protocol for communication of time critical information.

SEG.ACK
segment acknowledgment

SEG.LEN
segment length

SEG.PRC
segment precedence value

SEG.SEQ
segment sequence

SEG.UP
segment urgent pointer field

SEG.WND segment window field

segment A logical unit of data, in particular a TCP segment is the unit of data transferred between a pair of TCP modules.

segment acknowledgment The sequence number in the acknowledgment field of the arriving segment.

segment length The amount of sequence number space occupied by a segment, including any controls which occupy sequence space.

segment sequence The number in the sequence field of the arriving segment.

send sequence This is the next sequence number the local (sending) TCP will use on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.

send window This represents the sequence numbers which the remote (receiving) TCP is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP. The range of new sequence numbers which may be emitted by a TCP lies between SND.NXT and $\text{SND.UNA} + \text{SND.WND} - 1$. (Retransmissions of sequence numbers between SND.UNA and SND.NXT are expected, of course.)

SND.NXT send sequence

SND.UNA left sequence

SND.UP send urgent pointer

SND.WL1 segment sequence number at last window update

SND.WL2 segment acknowledgment number at last window update

September 1981

Transmission Control Protocol
Glossary

SND.WND send window

socket An address which specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.

Source Address The source address, usually the network and host identifiers.

SYN A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCB Transmission control block, the data structure that records the state of a connection.

TCB.PRC The precedence of the connection.

TCP Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

TOS Type of Service, an Internet Protocol field.

Type of Service An Internet Protocol field which indicates the type of service for this internet fragment.

URG A control bit (urgent), occupying no sequence space, used to indicate that the receiving user should be notified to do urgent processing as long as there is data to be consumed with sequence numbers less than the value indicated in the urgent pointer.

urgent pointer A control field meaningful only when the URG bit is on. This field communicates the value of the urgent pointer which indicates the data octet associated with the sending user's urgent call.

September 1981

Transmission Control Protocol

REFERENCES

- [1] Cerf, V., and R. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. COM-22, No. 5, pp 637-648, May 1974.
- [2] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.
- [3] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks, Vol. 2, No. 6, pp. 454-473, December 1978.
- [4] Postel, J., "Assigned Numbers", RFC 790, USC/Information Sciences Institute, September 1981.

APPLICATION LEVEL

TELNET PROTOCOL SPECIFICATION

FILE TRANSFER PROTOCOL

SIMPLE MAIL TRANSFER PROTOCOL

TRIVIAL FILE TRANSFER PROTOCOL

NAME SERVER PROTOCOL

RFC-764

TELNET PROTOCOL SPECIFICATION

June 1980

(223)

TELNET PROTOCOL SPECIFICATION

INTRODUCTION

The purpose of the TELNET Protocol is to provide a fairly general, bi-directional, eight-bit byte oriented communications facility. Its primary goal is to allow a standard method of interfacing terminal devices and terminal-oriented processes to each other. It is envisioned that the protocol may also be used for terminal-terminal communication ("linking") and process-process communication (distributed computation).

GENERAL CONSIDERATIONS

A TELNET connection is a Transmission Control Protocol (TCP) connection used to transmit data with interspersed TELNET control information. TCP and the connection establishment procedure are documented in the ARPA Internet Protocol Handbook.

The TELNET Protocol is built upon three main ideas: first, the concept of a "Network Virtual Terminal"; second, the principle of negotiated options; and third, a symmetric view of terminals and processes.

1. When a TELNET connection is first established, each end is assumed to originate and terminate at a "Network Virtual Terminal", or NVT. An NVT is an imaginary device which provides a standard, network-wide, intermediate representation of a canonical terminal. This eliminates the need for "server" and "user" Hosts* to keep information about the characteristics of each other's terminals and terminal handling conventions. All Hosts, both user and server, map their local device characteristics and conventions so as to appear to be dealing with an NVT over the network, and each can assume a similar mapping by the other party. The NVT is intended to strike a balance between being overly restricted (not providing Hosts a rich enough vocabulary for mapping into their local character sets), and being overly inclusive (penalizing users with modest terminals).

*NOTE: The "user" Host is the Host to which the physical terminal is normally attached, and the "server" host is the Host which is normally providing some service. As an alternate point of view, applicable even in terminal-to-terminal or process-to-process communications, the "user" Host is the Host which initiated the communication.

2. The principle of negotiated options takes cognizance of the fact that many sites will wish to provide additional services over and above those available within an NVT, and many users will have sophisticated terminals and would like to have elegant, rather than minimal, services. Independent of, but structured within, the TELNET Protocol various "options" will be sanctioned which can be used with the "DO, DON'T, WILL, WON'T" structure (discussed below) to allow a user and server to agree to use a more elaborate (or perhaps just different) set of conventions for their TELNET connection. Such options could include changing the character set, the echo mode, the line width, the page length, etc.

The basic strategy for setting up the use of options is to have either party (or both) initiate a request that some option take effect. The other party may then either accept or reject the request. If the request is accepted the option immediately takes effect; if it is rejected the associated aspect of the connection remains as specified for an NVT. Clearly, a party may always refuse a request to enable, and must never refuse a request to disable, some option since all parties must be prepared to support the NVT.

The syntax of option negotiation has been set up so that if both parties request an option simultaneously, each will see the other's request as the positive acknowledgment of its own.

3. The symmetry of the negotiation syntax can potentially lead to nonterminating acknowledgment loops -- each party seeing the incoming commands not as acknowledgments but as new requests which must be acknowledged. To prevent such loops, the following rules prevail:

- a. Parties may only request a change in option status; i.e., a party may not send out a "request" merely to announce what mode it is in.
- b. If a party receives what appears to be a request to enter some mode it is already in, the request should not be acknowledged.
- c. Whenever one party sends an option command to a second party, whether as a request or an acknowledgment, and use of the option will have any effect on the processing of the data being sent from the first party to the second, then the command must be inserted in the data stream at the point where it is desired that it take effect. (It should be noted that some time will elapse between the transmission of a request and the receipt of an acknowledgment, which may be negative. Thus, a site may wish to buffer data, after requesting an

option, until it learns whether the request is accepted or rejected, in order to hide the "uncertainty period" from the user.)

Option requests are likely to flurry back and forth when a TELNET connection is first established, as each party attempts to get the best possible service from the other party. Beyond that, however, options can be used to dynamically modify the characteristics of the connection to suit changing local conditions. For example, the NVT, as will be explained later, uses a transmission discipline well suited to the many "line at a time" applications such as BASIC, but poorly suited to the many "character at a time" applications such as NLS. A server might elect to devote the extra processor overhead required for a "character at a time" discipline when it was suitable for the local process and would negotiate an appropriate option. However, rather than then being permanently burdened with the extra processing overhead, it could switch (i.e., negotiate) back to NVT when the more "taut" control was no longer necessary.

It is possible for requests initiated by processes to stimulate a nonterminating request loop if the process responds to a rejection by merely re-requesting the option. To prevent such loops from occurring, rejected requests should not be repeated until something changes. Operationally, this can mean the process is running a different program, or the user has given another command, or whatever makes sense in the context of the given process and the given option. A good rule of thumb is that a re-request should only occur as a result of subsequent information from the other end of the connection or when demanded by local human intervention.

Option designers should not feel constrained by the somewhat limited syntax available for option negotiation. The intent of the simple syntax is to make it easy to have options--since it is correspondingly easy to profess ignorance about them. If some particular option requires a richer negotiation structure than possible within "DO, DON'T, WILL, WON'T", the proper tack is to use "DO, DON'T, WILL, WON'T" to establish that both parties understand the option, and once this is accomplished a more exotic syntax can be used freely. For example, a party might send a request to alter (establish) line length. If it is accepted, then a different syntax can be used for actually negotiating the line length--such a "sub-negotiation" perhaps including fields for minimum allowable, maximum allowable and desired line lengths. The important concept is that such expanded negotiations should never begin until some prior (standard) negotiation has established that both parties are capable of parsing the expanded syntax.

In summary, WILL XXX is sent, by either party, to indicate that party's desire (offer) to begin performing option XXX, DO XXX and DON'T XXX being its positive and negative acknowledgments; similarly, DO XXX is sent to indicate a desire (request) that the other party (i.e., the recipient of the DO) begin performing option XXX, WILL XXX and WON'T XXX being the positive and negative acknowledgments. Since the NVT is what is left when no options are enabled, the DON'T and WON'T responses are guaranteed to leave the connection in a state which both ends can handle. Thus, all Hosts may implement their TELNET processes to be totally unaware of options that are not supported, simply returning a rejection to (i.e., refusing) any option request that cannot be understood.

As much as possible, the TELNET protocol has been made server-user symmetrical so that it easily and naturally covers the user-user (linking) and server-server (cooperating processes) cases. It is hoped, but not absolutely required, that options will further this intent. In any case, it is explicitly acknowledged that symmetry is an operating principle rather than an ironclad rule.

A companion document, "TELNET Option Specifications," should be consulted for information about the procedure for establishing new options. That document, as well as descriptions of all currently defined options, is contained in the TELNET section of the ARPA Internet Protocol Handbook.

THE NETWORK VIRTUAL TERMINAL

The Network Virtual Terminal (NVT) is a bi-directional character device. The NVT has a printer and a keyboard. The printer responds to incoming data and the keyboard produces outgoing data which is sent over the TELNET connection and, if "echoes" are desired, to the NVT's printer as well. "Echoes" will not be expected to traverse the network (although options exist to enable a "remote" echoing mode of operation, no Host is required to implement this option). The code set is seven-bit USASCII in an eight-bit field, except as modified herein. Any code conversion and timing considerations are local problems and do not affect the NVT.

TRANSMISSION OF DATA

Although a TELNET connection through the network is intrinsically full duplex, the NVT is to be viewed as a half-duplex device operating in a line-buffered mode. That is, unless and until

options are negotiated to the contrary, the following default conditions pertain to the transmission of data over the TELNET connection:

- 1) Insofar as the availability of local buffer space permits, data should be accumulated in the Host where it is generated until a complete line of data is ready for transmission, or until some locally-defined explicit signal to transmit occurs. This signal could be generated either by a process or by a human user.

The motivation for this rule is the high cost, to some Hosts, of processing network input interrupts, coupled with the default NVT specification that "echoes" do not traverse the network. Thus, it is reasonable to buffer some amount of data at its source. Many systems take some processing action at the end of each input line (even line printers or card punches frequently tend to work this way), so the transmission should be triggered at the end of a line. On the other hand, a user or process may sometimes find it necessary or desirable to provide data which does not terminate at the end of a line; therefore implementers are cautioned to provide methods of locally signaling that all buffered data should be transmitted immediately.

- 2) When a process has completed sending data to an NVT printer and has no queued input from the NVT keyboard for further processing (i.e., when a process at one end of a TELNET connection cannot proceed without input from the other end), the process must transmit the TELNET Go Ahead (GA) command.

This rule is not intended to require that the TELNET GA command be sent from a terminal at the end of each line, since server Hosts do not normally require a special signal (in addition to end-of-line or other locally-defined characters) in order to commence processing. Rather, the TELNET GA is designed to help a user's local Host operate a physically half duplex terminal which has a "lockable" keyboard such as the IBM 2741. A description of this type of terminal may help to explain the proper use of the GA command.

The terminal-computer connection is always under control of either the user or the computer. Neither can unilaterally seize control from the other; rather the controlling end must relinquish its control explicitly. At the terminal end, the hardware is constructed so as to relinquish control each time

that a "line" is terminated (i.e., when the "New Line" key is typed by the user). When this occurs, the attached (local) computer processes the input data, decides if output should be generated, and if not returns control to the terminal. If output should be generated, control is retained by the computer until all output has been transmitted.

The difficulties of using this type of terminal through the network should be obvious. The "local" computer is no longer able to decide whether to retain control after seeing an end-of-line signal or not; this decision can only be made by the "remote" computer which is processing the data. Therefore, the TELNET GA command provides a mechanism whereby the "remote" (server) computer can signal the "local" (user) computer that it is time to pass control to the user of the terminal. It should be transmitted at those times, and only at those times, when the user should be given control of the terminal. Note that premature transmission of the GA command may result in the blocking of output, since the user is likely to assume that the transmitting system has paused, and therefore he will fail to turn the line around manually.

The foregoing, of course, does not apply to the user-to-server direction of communication. In this direction, GAs may be sent at any time, but need not ever be sent. Also, if the TELNET connection is being used for process-to-process communication, GAs need not be sent in either direction. Finally, for terminal-to-terminal communication, GAs may be required in neither, one, or both directions. If a Host plans to support terminal-to-terminal communication it is suggested that the Host provide the user with a means of manually signaling that it is time for a GA to be sent over the TELNET connection; this, however, is not a requirement on the implementer of a TELNET process.

STANDARD REPRESENTATION OF CONTROL FUNCTIONS

As stated in the Introduction to this document, the primary goal of the TELNET protocol is the provision of a standard interfacing of terminal devices and terminal-oriented processes through the network. Early experiences with this type of interconnection have shown that certain functions are implemented by most servers, but that the methods of invoking these functions differ widely. For a human user who interacts with several server systems, these differences are highly frustrating. TELNET, therefore, defines a standard representation for five of these functions, as described

below. These standard representations have standard, but not required, meanings (with the exception that the IP function may be required by other protocols which use TELNET); that is, a system which does not provide the function to local users need not provide it to network users and may treat the standard representation for the function as a No-operation. On the other hand, a system which does provide the function to local is obliged to provide the same function as a network user who transmits the standard representation for the function.

Interrupt Process (IP)

Many systems provide a function which suspends, interrupts, aborts, or terminates the operation of a user process. This function is frequently used when a user believes his process is in an unending loop, or when an unwanted process has been inadvertently activated. IP is the standard representation for invoking this function. It should be noted by implementers that IP may be required by other protocols which use TELNET, and therefore should be implemented if these other protocols are to be supported.

Abort Output (AC)

Many systems provide a function which allows a process, which is generating output, to run to completion (or to reach the same stopping point it would reach if running to completion) but without sending the output to the user's terminal. Further, this function typically clears any output already produced but not yet actually printed (or displayed) on the user's terminal. AO is the standard representation for invoking this function. For example, some subsystem might normally accept a user's command, send a long text string to the user's terminal in response, and finally signal readiness to accept the next command by sending a "prompt" character (preceded by <CR><LF>) to the user's terminal. If the AO were received during the transmission of the text string, a reasonable implementation would be to suppress the remainder of the text string, but transmit the prompt character and the preceding <CR><LF>. (This is possibly in distinction to the action which might be taken if an IP were received; the IP might cause suppression of the text string and an exit from the subsystem.)

It should be noted, by systems which provide this function, that there may be buffers external to the system (in the

network and the user's "local" Host) which should be cleared; the appropriate way to do this is to transmit the "Synch" signal described below.

Are You There (AYT)

Many systems provide a function which provides the user with some visible (e.g., printable) evidence that the system is still up and running. This function may be invoked by the user when the system is unexpectedly "silent" for a long time, because of the unanticipated (by the user) length of a computation, an unusually heavy system load, etc. AYT is the standard representation for invoking this function.

Erase Character (EC)

Many systems provide a function which deletes the last preceding undeleted character or "print position"* from the stream of data being supplied by the user. This function is typically used to edit keyboard input when typing mistakes are made. EC is the standard representation for invoking this function.

*NOTE: A "print position" may contain several characters which are the result of overstrikes, or of sequences such as <char1> BS <char2>...

Erase Line (EL)

Many systems provide a function which deletes all the data in the current "line" of input. This function is typically used to edit keyboard input. EL is the standard representation for invoking this function.

THE TELNET "SYNCH" SIGNAL

Most time-sharing systems provide mechanisms which allow a terminal user to regain control of a "runaway" process; the IP and AO functions described above are examples of these mechanisms. Such systems, when used locally, have access to all of the signals supplied by the user, whether these are normal characters or special "out of band" signals such as those supplied by the teletype "BREAK" key or the IBM 2741 "ATTN" key. This is not necessarily true when terminals are connected to the system

through the network; the network's flow control mechanisms may cause such a signal to be buffered elsewhere, for example in the user's Host.

To counter this problem, the TELNET "Synch" mechanism is introduced. A Synch signal consists of a TCP Urgent notification, coupled with the TELNET command DATA MARK. The Urgent notification, which is not subject to the flow control pertaining to the TELNET connection, is used to invoke special handling of the data stream by the process which receives it. In this mode, the data stream is immediately scanned for "interesting" signals as defined below, discarding intervening data. The TELNET command DATA MARK (DM) is the synchronizing mark in the data stream which indicates that any special signal has already occurred and the recipient can return to normal processing of the data stream.

The Synch is sent via the TCP send operation with the Urgent flag set and the DM as the last (or only) data octet.

When several Synchs are sent in rapid succession, the Urgent notifications may be merged. It is not possible to count Urgents since the number received will be less than or equal the number sent. When in normal mode a DM is a no operation, when in urgent mode it signals the end of the urgent processing (this should correspond with the end of Urgent pointer indicated by TCP).

If TCP indicates the end of Urgent data before the DM is found, TELNET should continue the special handling of the data stream until the DM is found.

"Interesting" signals are defined to be: the TELNET standard representations of IP, AO, and AYT (but not EC or EL); the local analogs of these standard representations (if any); all other TELNET commands; other site-defined signals which can be acted on without delaying the scan of the data stream.

Since one effect of the SYNCH mechanism is the discarding of essentially all characters (except TELNET commands) between the sender of the Synch and its recipient, this mechanism is specified as the standard way to clear the data path when that is desired. For example, if a user at a terminal causes an AO to be transmitted, the server which receives the AO (if it provides that function at all) should return a Synch to the user.

Finally, just as the TCP Urgent notification is needed at the

TELNET level as an out-of-band signal, so other protocols which make use of TELNET may require a TELNET command which can be viewed as an out-of-band signal at a different level.

By convention the sequence [IP, Synch] is to be used as such a signal. For example, suppose that some other protocol, which uses TELNET, defines the character string STOP analogously to the TELNET command AO. Imagine that a user of this protocol wishes a server to process the STOP string, but the connection is blocked because the server is processing other commands. The user should instruct his system to:

1. Send the TELNET IP character;
2. Send the TELNET SYNC sequence, that is:
 Send the Data Mark (DM) as the only character
 in a TCP urgent mode send operation.
3. Send the character string STOP; and
4. Send the other protocol's analog of the TELNET DM, if any.

The user (or process acting on his behalf) must transmit the TELNET SYNCH sequence of step 2 above to ensure that the TELNET IP gets through to the server's TELNET interpreter.

The Urgent should wake up the TELNET process, the IP should wake up the next higher level process.

THE NVT PRINTER AND KEYBOARD

The NVT printer has an unspecified carriage width and page length and can produce representations of all 95 USASCII graphics (codes 32 through 126). Of the 33 USASCII control codes (0 through 31 and 127), and the 128 uncovered codes (128 through 255), the following have specified meaning to the NVT printer:

NAME	CODE	MEANING
NULL (NUL)	0	A no operation
Line Feed (LF)	10	Moves the printer to the next print line, keeping the same horizontal position
Carriage Return (CR)	13	Moves the printer to the left margin of the current line.

In addition, the following codes shall have defined, but not required, effects on the NVT printer. Neither end of a TELNET connection may assume that the other party will take, or will have taken, any particular action upon receipt or transmission of these:

BELL (BEL)	7	Produces an audible or visible signal (which does NOT move the print head)
Back Space (BS)	8	Moves the print head one character position towards the left margin.
Horizontal Tab (HT)	9	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Vertical Tab (VT)	11	Moves the printer to the next horizontal tab stop. It remains unspecified how either party determines or establishes where such tab stops are located.
Form Feed (FF)	12	Moves the printer to the top of the next page, keeping the same horizontal position

All remaining codes do not cause the NVT printer to take any action.

The sequence "CR LF", as defined, will cause the NVT to be positioned at the left margin of the next print line (as would, for example, the sequence "LF CR"). However, many systems and terminals do not treat CR and LF independently, and will have to go to some effort to simulate their effect. (For example, some terminals do not have a CR independent of the LF, but on such terminals it may be possible to simulate a CR by backspacing.) Therefore, the sequence "CR LF" must be treated as a single "new line" character and used whenever their combined action is intended; the sequence "CR NUL" must be used where a carriage return alone is actually desired; and the CR character must be avoided in other contexts. This rule gives assurance to systems which must decide whether to perform a "new line" function or a multiple-backspace that the TELNET stream contains a character following a CR that will allow a rational decision.

Note that "CR LF" or "CR NUL" is required in both directions (in the default ASCII mode), to preserve the symmetry of the NVT model. Even though it may be known in some situations (e.g., with remote echo and suppress go ahead options in effect) that characters are not being sent to an actual printer, none the less, for the sake of consistency, the protocol requires that a NUL be inserted following a CR not followed by a LF in the data stream. The converse of this is that a NUL received in the data stream after a CR (in the absence of options negotiations which explicitly specify otherwise) should be stripped out prior to applying the NVT to local character set mapping.

The NVT keyboard has keys, or key combinations, or key sequences, for generating all 128 USASCII codes. Note that although many have no effect on the NVT printer, the NVT keyboard is capable of generating them.

In addition to these codes, the NVT keyboard shall be capable of generating the following additional codes which, except as noted, have defined, but not required, meanings. The actual code assignments for these "characters" are in the TELNET Command section, because they are viewed as being, in some sense, generic and should be available even when the data stream is interpreted as being some other character set.

Synch

This key allows the user to clear his data path to the other party. The activation of this key causes a DM (see command section) to be sent in the data stream and a TCP Urgent notification is associated with it. The pair DM-Urgent is to have required meaning as defined previously.

Break (BRK)

This code is provided because it is a signal outside the USASCII set which is currently given local meaning within many systems. It is intended to indicate that the Break Key or the Attention Key was hit. Note, however, that this is intended to provide a 129th code for systems which require it, not as a synonym for the IP standard representation.

Interrupt Process (IP)

Suspend, interrupt, abort or terminate the process to which the NVT is connected. Also, part of the out-of-band signal for other protocols which use TELNET.

Abort Output (AO)

Allow the current process to (appear to) run to completion, but do not send its output to the user. Also, send a Synch to the user.

Are You There (AYT)

Send back to the NVT some visible (i.e., printable) evidence that the AYT was received.

Erase Character (EC)

The recipient should delete the last preceding undeleted character or "print position" from the data stream.

Erase Line (EL)

The recipient should delete characters from the data stream back to, but not including, the last "CR LF" sequence sent over the TELNET connection.

The spirit of these "extra" keys, and also the printer format effectors, is that they should represent a natural extension of the mapping that already must be done from "NVT" into "local". Just as the NVT data byte 104 should be mapped into whatever the local code for "uppercase D" is, so the EC character should be mapped into whatever the local "Erase Character" function is. Further, just as the mapping for 174 is somewhat arbitrary in an environment that has no "vertical bar" character, the EL character may have a somewhat arbitrary mapping (or none at all) if there is no local "Erase Line" facility. Similarly for format effectors: if the terminal actually does have a "Vertical tab", then the mapping for VT is obvious, and only when the terminal does not have a vertical tab should the effect of VT be unpredictable.

TELNET COMMAND STRUCTURE

All TELNET commands consist of at least a two byte sequence: the "Interpret as Command" (IAC) escape character followed by the code for the command. The commands dealing with option negotiation are three byte sequences, the third byte being the code for the option referenced. This format was chosen so that as more comprehensive use of the "data space" is made -- by negotiations from the basic NVT, of course -- collisions of data bytes with reserved command values will be minimized, all such collisions requiring the inconvenience, and inefficiency, of "escaping" the data bytes into the stream. With the current set-up, only the IAC need be doubled to be sent as data, and the other 255 codes may be passed transparently.

The following are the defined TELNET commands. Note that these codes and code sequences have the indicated meaning only when immediately preceded by an IAC.

NAME	CODE	MEANING
SE	240	End of subnegotiation parameters
NOP	241	No operation
Data Mark	242	The data stream portion of a Synch This should always be accompanied by a TCP Urgent notification.
Break	243	NVT character BRK
Interrupt Process	244	The function IP
Abort output	245	The function AO
Are You There	246	The function AYT
Erase character	247	The function EC
Erase Line	248	The function EL
Go ahead	249	The GA signal
SB	250	Indicates that what follows is subnegotiation of the indicated option
WILL (option code)	251	Indicates the desire to begin performing, or confirmation that you are now performing, the indicated option
WON't (option code)	252	Indicates the refusal to perform, or continue performing, the indicated option.
DO (option code)	253	Indicates the request that the other party perform, or confirmation that you are expecting the other party to perform, the

indicated option.
DON'T (option code) 254 Indicates the demand that the
other party stop performing,
or confirmation that you are no
longer expecting the other party
to perform, the indicated option.
IAC 255 Data Byte 255.

CONNECTION ESTABLISHMENT

The TELNET TCP connection is established between the user's port U and the server's port L. The server listens on its well known port L for such connections. Since a TCP connection is full duplex and identified by the pair of ports, the server can engage in many simultaneous connections involving it's port L and different user ports U.

Port Assignment

When used for remote user access to service hosts (i.e., remote terminal access) this protocol is assigned server port 23 (27 octal). That is L=23.

RFC-765

FILE TRANSFER PROTOCOL

June 1980

(241)

FILE TRANSFER PROTOCOL

INTRODUCTION

The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among Hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs.

The attempt in this specification is to satisfy the diverse needs of users of maxi-Hosts, mini-Hosts, and TIPS, with a simple, and easily implemented protocol design.

This paper assumes knowledge of the following protocols described in the ARPA Internet Protocol Handbook.

The Transmission Control Protocol

The TELNET Protocol

DISCUSSION

In this section, the terminology and the FTP model are discussed. The terms defined in this section are only those that have special significance in FTP. Some of the terminology is very specific to the FTP model; some readers may wish to turn to the section on the FTP model while reviewing the terminology.

TERMINOLOGY

ASCII

The ASCII character set as defined in the ARPA Internet Protocol Handbook. In FTP, ASCII characters are defined to be the lower half of an eight-bit code set (i.e., the most significant bit is zero).

access controls

Access controls define users' access privileges to the use of a system, and to the files in that system. Access controls are necessary to prevent unauthorized or accidental use of files. It is the prerogative of a server-FTP process to invoke access controls.

byte size

There are two byte sizes of interest in FTP: the logical byte size of the file, and the transfer byte size used for the transmission of the data. The transfer byte size is always 8 bits. The transfer byte size is not necessarily the byte size in which data is to be stored in a system, nor the logical byte size for interpretation of the structure of the data.

data connection

A simplex connection over which data is transferred, in a specified mode and type. The data transferred may be a part of a file, an entire file or a number of files. The path may be between a server-DTP and a user-DTP, or between two server-DTPs.

data port

The passive data transfer process "listens" on the data port for a connection from the active transfer process in order to open the data connection.

EOF

The end-of-file condition that defines the end of a file being transferred.

EOR

The end-of-record condition that defines the end of a record being transferred.

error recovery

A procedure that allows a user to recover from certain errors such as failure of either Host system or transfer process. In FTP, error recovery may involve restarting a file transfer at a given checkpoint.

FTP commands

A set of commands that comprise the control information flowing from the user-FTP to the server-FTP process.

file

An ordered set of computer data (including programs), of arbitrary length, uniquely identified by a pathname.

mode

The mode in which data is to be transferred via the data connection. The mode defines the data format during transfer including EOR and EOF. The transfer modes defined in FTP are described in the Section on Transmission Modes.

NVT

The Network Virtual Terminal as defined in the TELNET Protocol.

NVFS

The Network Virtual File System. A concept which defines a standard network file system with standard commands and pathname conventions. FTP only partially implements the NVFS concept at this time.

page

A file may be structured as a set of independent parts called pages. FTP supports the transmission of discontinuous files as independent indexed pages.

pathname

Pathname is defined to be the character string which must be input to a file system by a user in order to identify a file. Pathname normally contains device and/or directory names, and file name specification. FTP does not yet specify a standard pathname convention. Each user must follow the file naming conventions of the file systems involved in the transfer.

record

A sequential file may be structured as a number of contiguous parts called records. Record structures are supported by FTP but a file need not have record structure.

reply

A reply is an acknowledgment (positive or negative) sent from server to user via the TELNET connections in response to FTP commands. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

server-DTP

The data transfer process, in its normal "active" state, establishes the data connection with the "listening" data port, sets up parameters for transfer and storage, and transfers data on command from its PI. The DTP can be placed in a "passive" state to listen for, rather than initiate a, connection on the data port.

server-FTP process

A process or set of processes which perform the function of file transfer in cooperation with a user-FTP process and, possibly, another server. The functions consist of a protocol interpreter (PI) and a data transfer process (DTP).

server-PI

The protocol interpreter "listens" on Port L for a connection from a user-PI and establishes a TELNET communication connection. It receives standard FTP commands from the user-PI, sends replies, and governs the server-DTP.

TELNET connections

The full-duplex communication path between a user-PI and a server-PI, operating according to the TELNET Protocol.

type

The data representation type used for data transfer and storage. Type implies certain transformations between the time of data storage and data transfer. The representation types defined in FTP are described in the Section on Establishing Data Connections.

user

A human being or a process on behalf of a human being wishing to obtain file transfer service. The human user may interact directly with a server-FTP process, but use of a user-FTP process is preferred since the protocol design is weighted towards automata.

user-DTP

The data transfer process "listens" on the data port for a connection from a server-FTP process. If two servers are transferring data between them, the user-DTP is inactive.

user-FTP process

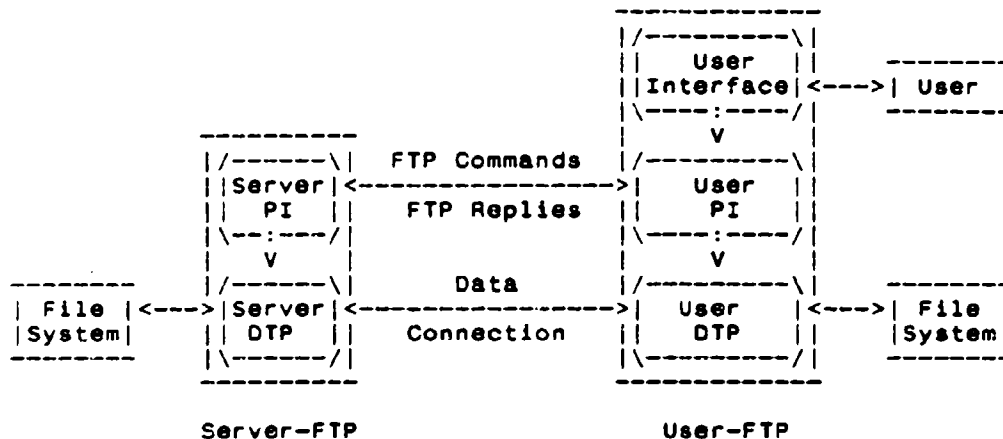
A set of functions including a protocol interpreter, a data transfer process and a user interface which together perform the function of file transfer in cooperation with one or more server-FTP processes. The user interface allows a local language to be used in the command-reply dialogue with the user.

user-PI

The protocol interpreter initiates the TELNET connection from its port U to the server-FTP process, initiates FTP commands, and governs the user-DTP if that process is part of the file transfer.

THE FTP MODEL

With the above definitions in mind, the following model (shown in Figure 1) may be diagrammed for an FTP service.



- NOTES: 1. The data connection may be used in either direction.
2. The data connection need not exist all of the time.

Figure 1 Model for FTP Use

In the model described in Figure 1, the user-protocol interpreter initiates the TELNET connection. At the initiation of the user, standard FTP commands are generated by the user-PI and transmitted to the server process via the TELNET connection. (The user may establish a direct TELNET connection to the server-FTP, from a TIP terminal for example, and generate standard FTP commands himself, bypassing the user-FTP process.) Standard replies are sent from the server-PI to the user-PI over the TELNET connection in response to the commands.

The FTP commands specify the parameters for the data connection (data port, transfer mode, representation type, and structure) and the nature of file system operation (store, retrieve, append, delete, etc.). The user-DTP or its designate should "listen" on the specified data port, and the server initiate the data connection and data transfer in accordance with the specified parameters. It should be noted that the data port need not be in

the same Host that initiates the FTP commands via the TELNET connection, but the user or his user-FTP process must ensure a "listen" on the specified data port. It should also be noted that the data connection may be used for simultaneous sending and receiving.

In another situation a user might wish to transfer files between two Hosts, neither of which is his local Host. He sets up TELNET connections to the two servers and then arranges for a data connection between them. In this manner control informatio is passed to the user-PI but data is transferred between the server data transfer processes. Following is a model of this server-server interaction.

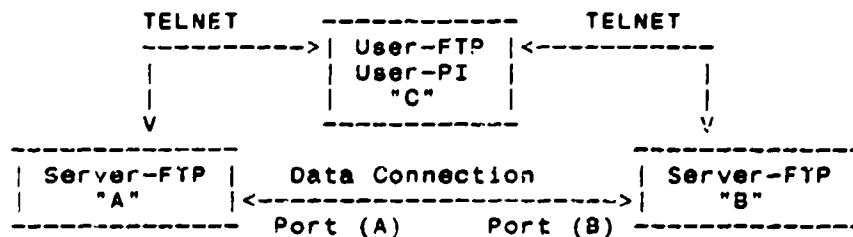


Figure 2

The protocol requires that the TELNET connections be open while data transfer is in progress. It is the responsibility of the user to request the closing of the TELNET connections when finished using the FTP service, while it is the server who takes the action. The server may abort data transfer if the TELNET connections are closed without command.

DATA TRANSFER FUNCTIONS

Files are transferred only via the data connection. The TELNET connection is used for the transfer of commands, which describe the functions to be performed, and the replies to these commands (see the Section on FTP Replies). Several commands are concerned with the transfer of data between Hosts. These data transfer commands include the MODE command which specify how the bits of the data are to be transmitted, and the STRUcture and TYPE commands, which are used to define the way in which the data are to be represented. The transmission and representation are basically independent but

"Stream" transmission mode is dependent on the file structure attribute and if "Compressed" transmission mode is used the nature of the filler byte depends on the representation type.

DATA REPRESENTATION AND STORAGE

Data is transferred from a storage device in the sending Host to a storage device in the receiving Host. Often it is necessary to perform certain transformations on the data because data storage representations in the two systems are different. For example, NVT-ASCII has different data storage representations in different systems. PDP-10's generally store NVT-ASCII as five 7-bit ASCII characters, left-justified in a 36-bit word. 360's store NVT-ASCII as 8-bit EBCDIC codes. Multics stores NVT-ASCII as four 9-bit characters in a 36-bit word. It may be desirable to convert characters into the standard NVT-ASCII representation when transmitting text between dissimilar systems. The sending and receiving sites would have to perform the necessary transformations between the standard representation and their internal representations.

A different problem in representation arises when transmitting binary data (not character codes) between Host systems with different word lengths. It is not always clear how the sender should send data, and the receiver store it. For example, when transmitting 32-bit bytes from a 32-bit word-length system to a 36-bit word-length system, it may be desirable (for reasons of efficiency and usefulness) to store the 32-bit bytes right-justified in a 36-bit word in the latter system. In any case, the user should have the option of specifying data representation and transformation functions. It should be noted that FTP provides for very limited data type representations. Transformations desired beyond this limited capability should be performed by the user directly.

Data representations are handled in FTP by a user specifying a representation type. This type may implicitly (as in ASCII or EBCDIC) or explicitly (as in local byte) define a byte size for interpretation which is referred to as the "logical byte size." This has nothing to do with the byte size used for transmission over the data connection, called the "transfer byte size", and the two should not be confused. For example, NVT-ASCII has a logical byte size of 8 bits. If the type is local byte, then the TYPE command has an obligatory second parameter specifying the logical byte size. The transfer byte size is always 8 bits.

The types ASCII and EBCDIC also take a second (optional) parameter; this is to indicate what kind of vertical format control, if any, is associated with a file. The following data representation types are defined in FTP:

ASCII Format

This is the default type and must be accepted by all FTP implementations. It is intended primarily for the transfer of text files, except when both Hosts would find the EBCDIC type more convenient.

The sender converts the data from his internal character representation to the standard 8-bit NVT-ASCII representation (see the TELNET specification). The receiver will convert the data from the standard form to his own internal form.

In accordance with the NVT standard, the <CRLF> sequence should be used, where necessary, to denote the end of a line of text. (See the discussion of file structure at the end of the Section on Data Representation and Storage).

Using the standard NVT-ASCII representation means that data must be interpreted as 8-bit bytes.

The Format parameter for ASCII and EBCDIC types is discussed below.

EBCDIC Format

This type is intended for efficient transfer between Hosts which use EBCDIC for their internal character representation.

For transmission the data are represented as 8-bit EBCDIC characters. The character code is the only difference between the functional specifications of EBCDIC and ASCII types.

End-of-line (as opposed to end-of-record--see the discussion of structure) will probably be rarely used with EBCDIC type for purposes of denoting structure, but where it is necessary the <NL> character should be used.

A character file may be transferred to a Host for one of three purposes: for printing, for storage and later retrieval, or for processing. If a file is sent for printing, the receiving Host must know how the vertical format control is represented. In the second case, it must be possible to store a file at a Host and then retrieve it later in exactly the same form. Finally, it ought to be possible to move a file from one Host to another and process the file at the second Host without undue trouble. A single ASCII or EBCDIC format does not satisfy all these conditions and so these types have a second parameter specifying one of the following three formats:

Non-print

This is the default format to be used if the second (format) parameter is omitted. Non-print format must be accepted by all FTP implementations.

The file need contain no vertical format information. If it is passed to a printer process, this process may assume standard values for spacing and margins.

Normally, this format will be used with files destined for processing or just storage.

TELNET Format Controls

The file contains ASCII/EBCDIC vertical format controls (i.e., <CR>, <LF>, <NL>, <VT>, <FF>) which the printer process will interpret appropriately. <CRLF>, in exactly this sequence, also denotes end-of-line.

Carriage Control (ASA)

The file contains ASA (FORTRAN) vertical format control characters. (See RFC 740 Appendix C and Communications of the ACM, Vol. 7, No. 10, 606 (Oct. 1964)). In a line or a record, formatted according to the ASA Standard, the first character is not to be printed. Instead it should be used to determine the vertical movement of the paper which should take place before the rest of the record is printed.

The ASA Standard specifies the following control characters:

Character	Vertical Spacing
blank	Move paper up one line
0	Move paper up two lines
1	Move paper to top of next page
+	No movement, i.e., overprint

Clearly there must be some way for a printer process to distinguish the end of the structural entity. If a file has record structure (see below) this is no problem; records will be explicitly marked during transfer and storage. If the file has no record structure, the <CRLF> end-of-line sequence is used to separate printing lines, but these format effectors are overridden by the ASA controls.

Image

The data are sent as contiguous bits which, for transfer, are packed into the 8-bit transfer bytes. The receiving site must store the data as contiguous bits. The structure of the storage system might necessitate the padding of the file (or of each record, for a record-structured file) to some convenient boundary (byte, word or block). This padding, which must be all zeros, may occur only at the end of the file (or at the end of each record) and there must be a way of identifying the padding bits so that they may be stripped off if the file is retrieved. The padding transformation should be well publicized to enable a user to process a file at the storage site.

Image type is intended for the efficient storage and retrieval of files and for the transfer of binary data. It is recommended that this type be accepted by all FTP implementations.

Local byte size

The data is transferred in logical bytes of the size specified by the obligatory second parameter, Byte size. The value of Byte size must be a decimal integer; there is no default value. The logical byte size is not necessarily the same as the transfer byte size. If there is a

difference in byte sizes, then the logical bytes should be packed contiguously, disregarding transfer byte boundaries and with any necessary padding at the end.

When the data reaches the receiving Host it will be transformed in a manner dependent on the logical byte size and the particular Host. This transformation must be invertible (that is an identical file can be retrieved if the same parameters are used) and should be well publicized by the FTP implementors.

For example, a user sending 36-bit floating-point numbers to a Host with a 32-bit word could send his data as Local byte with a logical byte size of 36. The receiving Host would then be expected to store the logical bytes so that they could be easily manipulated; in this example putting the 36-bit logical bytes into 64-bit double words should suffice.

Another example, a pair of hosts with a 36-bit word size may send data to one another in words by using TYPE L 36. The data would be sent in the 8-bit transmission bytes packed so that 9 transmission bytes carried two host words.

A note of caution about parameters: a file must be stored and retrieved with the same parameters if the retrieved version is to be identical to the version originally transmitted. Conversely, FTP implementations must return a file identical to the original if the parameters used to store and retrieve a file are the same.

In addition to different representation types, FTP allows the structure of a file to be specified. Three file structures are defined in FTP:

- file-structure, where there is no internal structure and the file is considered to be a continuous sequence of data bytes,

- record-structure, where the file is made up of sequential records,

- and page-structure, where the file is made up of independent indexed pages.

File-structure is the default, to be assumed if the STRucture command has not been used but both file and record structures must

be accepted for "text" files (i.e., files with TYPE ASCII or EBCDIC) by all FTP implementations. The structure of a file will affect both the transfer mode of a file (see the Section on Transmission Modes) and the interpretation and storage of the file.

The "natural" structure of a file will depend on which Host stores the file. A source-code file will usually be stored on an IBM 360 in fixed length records but on a PDP-10 as a stream of characters partitioned into lines, for example by <CRLF>. If the transfer of files between such disparate sites is to be useful, there must be some way for one site to recognize the other's assumptions about the file.

With some sites being naturally file-oriented and others naturally record-oriented there may be problems if a file with one structure is sent to a Host oriented to the other. If a text file is sent with record-structure to a Host which is file oriented, then that Host should apply an internal transformation to the file based on the record structure. Obviously this transformation should be useful but it must also be invertible so that an identical file may be retrieved using record structure.

In the case of a file being sent with file-structure to a record-oriented Host, there exists the question of what criteria the Host should use to divide the file into records which can be processed locally. If this division is necessary the FTP implementation should use the end-of-line sequence, <CRLF> for ASCII, or <NL> for EBCDIC text files, as the delimiter. If an FTP implementation adopts this technique, it must be prepared to reverse the transformation if the file is retrieved with file-structure.

Page Structure

To transmit files that are discontinuous FTP defines a page structure. Files of this type are sometimes known as "random access files" or even as "holoy files". In these files there is sometimes other information associated with the file as a whole (e.g., a file descriptor), or with a section of the file (e.g., page access controls), or both. In FTP, the sections of the file are called pages.

To provide for various page sizes and associated information each page is sent with a page header. The page header has the following defined fields:

Header Length

The number of logical bytes in the page header including this byte. The minimum header length is 4.

Page Index

The logical page number of this section of the file. This is not the transmission sequence number of this page, but the index used to identify this page of the file.

Data Length

The number of logical bytes in the page data. The minimum data length is 0.

Page Type

The type of page this is. The following page types are defined:

0 = Last Page

This is used to indicate the end of a paged structured transmission. The header length must be 4, and the data length must be 0.

1 = Simple Page

This is the normal type for simple paged files with no page level associated control information. The header length must be 4.

2 = Descriptor Page

This type is used to transmit the descriptive information for the file as a whole.

3 = Access Controlled Page

This type includes an additional header field for paged files with page level access control information. The header length must be 5.

Optional Fields

Further header fields may be used to supply per page control information, for example, per page access control.

All fields are one logical byte in length. The logical byte size is specified by the TYPE command.

ESTABLISHING DATA CONNECTIONS

The mechanics of transferring data consists of setting up the data connection to the appropriate ports and choosing the parameters for transfer. Both the user and the server-DTPs have a default data port. The user-process default data port is the same as the control connection port, i.e., U. The server-process default data port is the port adjacent to the control connection port, i.e., L-1.

The transfer byte size is 8-bit bytes. This byte size is relevant only for the actual transfer of the data; it has no bearing on representation of the data within a Host's file system.

The passive data transfer process (this may be a user-DTP or a second server-DTP) shall "listen" on the data port prior to sending a transfer request command. The FTP request command determines the direction of the data transfer. The server, upon receiving the transfer request, will initiate the data connection to the port. When the connection is established, the data transfer begins between DTP's, and the server-PI sends a confirming reply to the user-PI.

It is possible for the user to specify an alternate data port by use of the PORT command. He might want a file dumped on a TYP line printer or retrieved from a third party Host. In the latter case the user-PI sets up TELNET connections with both server-PI's. One server is then told (by an FTP command) to "listen" for a connection which the other will initiate. The user-PI sends one server-PI a PORT command indicating the data port of the other. Finally both are sent the appropriate transfer commands. The exact sequence of commands and replies sent between the user-controller and the servers is defined in the Section on FTP Replies.

In general it is the server's responsibility to maintain the data connection--to initiate it and to close it. The exception to this

is when the user-DTP is sending the data in a transfer mode that requires the connection to be closed to indicate EOF. The server MUST close the data connection under the following conditions:

1. The server has completed sending data in a transfer mode that requires a close to indicate EOF.
2. The server receives an ABORT command from the user.
3. The port specification is changed by a command from the user.
4. The TELNET connection is closed legally or otherwise.
5. An irrecoverable error condition occurs.

Otherwise the close is a server option, the exercise of which he must indicate to the user-process by an appropriate reply.

TRANSMISSION MODES

The next consideration in transferring data is choosing the appropriate transmission mode. There are three modes: one which formats the data and allows for restart procedures; one which also compresses the data for efficient transfer; and one which passes the data with little or no processing. In this last case the mode interacts with the structure attribute to determine the type of processing. In the compressed mode the representation type determines the filler byte.

All data transfers must be completed with an end-of-file (EOF) which may be explicitly stated or implied by the closing of the data connection. For files with record structure, all the end-of-record markers (EOR) are explicit, including the final one. For files transmitted in page structure a "last-page" page type is used.

NOTE: In the rest of this section, byte means "transfer byte" except where explicitly stated otherwise.

For the purpose of standardized transfer, the sending Host will translate his internal end of line or end of record denotation into the representation prescribed by the transfer mode and file structure, and the receiving Host will perform the inverse translation to his internal denotation. An IBM 360 record count field may not be recognized at another Host, so the end of record

information may be transferred as a two byte control code in Stream mode or as a flagged bit in a Block or Compressed mode descriptor. End of line in an ASCII or EBCDIC file with no record structure should be indicated by <CRLF> or <NL>, respectively. Since these transformations imply extra work for some systems, identical systems transferring non-record structured text files might wish to use a binary representation and stream mode for the transfer.

The following transmission modes are defined in FTP:

STREAM

The data is transmitted as a stream of bytes. There is no restriction on the representation type used; record structures are allowed.

In a record structured file EOR and EOF will each be indicated by a two-byte control code. The first byte of the control code will be all ones, the escape character. The second byte will have the low order bit on and zeros elsewhere for EOR and the second low order bit on for EOF; that is, the byte will have value 1 for EOR and value 2 for EOF. EOR and EOF may be indicated together on the last byte transmitted by turning both low order bits on, i.e., the value 3. If a byte of all ones was intended to be sent as data, it should be repeated in the second byte of the control code.

If the structure is file structure, the EOF is indicated by the sending Host closing the data connection and all bytes are data bytes.

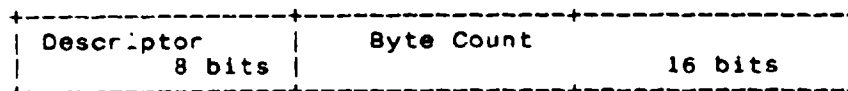
BLOCK

The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF) last block in the record (EOR), restart marker (see the Section on Error Recovery and Restart) or suspect data (i.e., the data being transferred is suspected of errors and is not reliable). This last code is NOT intended for error control within FTP. It is motivated by the desire of sites

exchanging certain types of data (e.g., seismic or weather data) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain portions are suspect). Record structures are allowed in this mode, and any representation type may be used.

The header consists of the three bytes. Of the 24 bits of header information, the 16 low order bits shall represent byte count, and the 8 high order bits shall represent descriptor codes as shown below.

Block Header



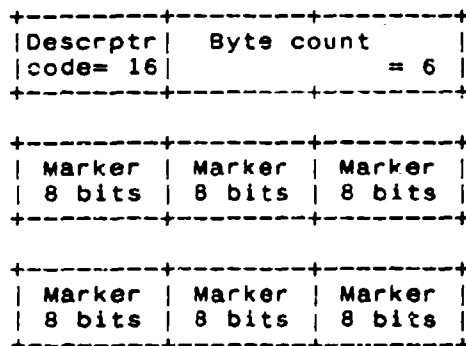
The descriptor codes are indicated by bit flags in the descriptor byte. Four codes have been assigned, where each code number is the decimal value of the corresponding bit in the byte.

Code	Meaning
128	End of data block is EOR
64	End of data block is EOF
32	Suspected errors in data block
16	Data block is a restart marker

With this encoding more than one descriptor coded condition may exist for a particular block. As many bits as necessary may be flagged.

The restart marker is embedded in the data stream as an integral number of 8-bit bytes representing printable characters in the language being used over the TELNET connection (e.g., default--NVT-ASCII). <SP> (Space, in the appropriate language) must not be used WITHIN a restart marker.

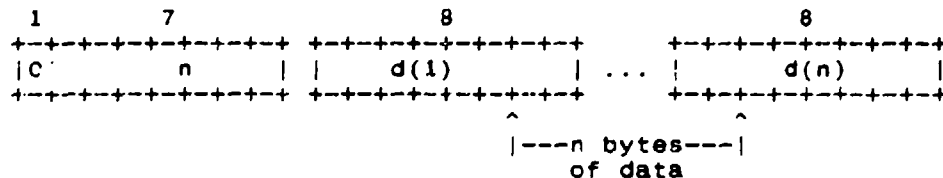
For example, to transmit a six-character marker, the following would be sent:



COMPRESSED

There are three kinds of information to be sent: regular data, sent in a byte string; compressed data, consisting of replications or filler; and control information, sent in a two-byte escape sequence. If $n > 0$ bytes (up to 127) of regular data are sent, these n bytes are preceded by a byte with the left-most bit set to 0 and the right-most 7 bits containing the number n .

Byte string:



String of n data bytes $d(1), \dots, d(n)$
 Count n must be positive.

To compress a string of n replications of the data byte d , the following 2 bytes are sent:

Replicated Byte:

```
      2      6      8
+---+---+---+---+---+---+---+---+
| 1 0 |   n   | |   d   |
+---+---+---+---+---+---+---+---+
```

A string of n filler bytes can be compressed into a single byte, where the filler byte varies with the representation type. If the type is ASCII or EBCDIC the filler byte is <SP> (Space, ASCII code 32., EBCDIC code 64). If the type is Image or Local byte the filler is a zero byte.

Filler String:

```
      2      6
+---+---+---+---+---+---+
| 1 1 |   n   |
+---+---+---+---+---+---+
```

The escape sequence is a double byte, the first of which is the escape byte (all zeros) and the second of which contains descriptor codes as defined in Block mode. The descriptor codes have the same meaning as in Block mode and apply to the succeeding string of bytes.

Compressed mode is useful for obtaining increased bandwidth on very large network transmissions at a little extra CPU cost. It can be most effectively used to reduce the size of printer files such as those generated by RJE Hosts.

ERROR RECOVERY AND RESTART

There is no provision for detecting bits lost or scrambled in data transfer; this level of error control is handled by the TCP. However, a restart procedure is provided to protect users from gross system failures (including failures of a Host, an FTP-process, or the underlying network).

The restart procedure is defined only for the block and compressed modes of data transfer. It requires the sender of data to insert a special marker code in the data stream with some marker information. The marker information has meaning only to the sender, but must consist of printable characters in the default or negotiated language of the TELNET connection (ASCII or EBCDIC). The marker could represent a bit-count, a record-count, or any

other information by which a system may identify a data checkpoint. The receiver of data, if it implements the restart procedure, would then mark the corresponding position of this marker in the receiving system, and return this information to the user.

In the event of a system failure, the user can restart the data transfer by identifying the marker point with the FTP restart procedure. The following example illustrates the use of the restart procedure.

The sender of the data inserts an appropriate marker block in the data stream at a convenient point. The receiving Host marks the corresponding data point in its file system and conveys the last known sender and receiver marker information to the user, either directly or over the TELNET connection in a 110 reply (depending on who is the sender). In the event of a system failure, the user or controller process restarts the server at the last server marker by sending a restart command with server's marker code as its argument. The restart command is transmitted over the TELNET connection and is immediately followed by the command (such as RETR, STOR or LIST) which was being executed when the system failure occurred.

FILE TRANSFER FUNCTIONS

The communication channel from the user-PI to the server-PI is established by a TCP connection from the user to a standard server port. The user protocol interpreter is responsible for sending FTP commands and interpreting the replies received; the server-PI interprets commands, sends replies and directs its DTP to set up the data connection and transfer the data. If the second party to the data transfer (the passive transfer process) is the user-DTP then it is governed through the internal protocol of the user-FTP Host; if it is a second server-DTP then it is governed by its PI on command from the user-PI. The FTP replies are discussed in the next section. In the description of a few of the commands in this section it is helpful to be explicit about the possible replies.

FTP COMMANDS

ACCESS CONTROL COMMANDS

The following commands specify access control identifiers (command codes are shown in parentheses).

USER NAME (USER)

The argument field is a TELNET string identifying the user. The user identification is that which is required by the server for access to its file system. This command will normally be the first command transmitted by the user after the TELNET connections are made (some servers may require this). Additional identification information in the form of a password and/or an account command may also be required by some servers. Servers may allow a new USER command to be entered at any point in order to change the access control and/or accounting information. This has the effect of flushing any user, password, and account information already supplied and beginning the login sequence again. All transfer parameters are unchanged and any file transfer in progress is completed under the old account.

PASSWORD (PASS)

The argument field is a TELNET string identifying the user's password. This command must be immediately preceded by the user name command, and, for some sites, completes the user's identification for access control. Since password information is quite sensitive, it is desirable in general to "mask" it or suppress typeout. It appears that the server has no foolproof way to achieve this. It is therefore the responsibility of the user-FTP process to hide the sensitive password information.

ACCOUNT (ACCT)

The argument field is a TELNET string identifying the user's account. The command is not necessarily related to the USER command, as some sites may require an account for login and others only for specific access, such as storing files. In the latter case the command may arrive at any time.

There are reply codes to differentiate these cases for the automaton: when account information is required for login, the response to a successful PASSWORD command is reply code 332. On the other hand, if account information is NOT required for login, the reply to a successful PASSWORD command is 230; and if the account information is needed for a command issued later in the dialogue, the server should

return a 332 or 532 reply depending on whether he stores (pending receipt of the ACCOUNT command) or discards the command, respectively.

REINITIALIZE (REIN)

This command terminates a USER, flushing all I/O and account information, except to allow any transfer in progress to be completed. All parameters are reset to the default settings and the TELNET connection is left open. This is identical to the state in which a user finds himself immediately after the TELNET connection is opened. A USER command may be expected to follow.

LOGOUT (QUIT)

This command terminates a USER and if file transfer is not in progress, the server closes the TELNET connection. If file transfer is in progress, the connection will remain open for result response and the server will then close it. If the user-process is transferring files for several USERS but does not wish to close and then reopen connections for each, then the REIN command should be used instead of QUIT.

An unexpected close on the TELNET connection will cause the server to take the effective action of an abort (ABOR) and a logout (QUIT).

TRANSFER PARAMETER COMMANDS

All data transfer parameters have default values, and the commands specifying data transfer parameters are required only if the default parameter values are to be changed. The default value is the last specified value, or if no value has been specified, the standard default value as stated here. This implies that the server must "remember" the applicable default values. The commands may be in any order except that they must precede the FTP service request. The following commands specify data transfer parameters.

DATA PORT (PORT)

The argument is a HOST-PORT specification for the data port to be used in data connection. There defaults for both the user and server data ports, and under normal circumstances this command and its reply are not needed. If this command

is used the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation). The fields are separated by commas. A port command would be:

PORT h1,h2,h3,h4,p1,p2

where, h1 is the high order 8 bits of the internet host address.

PASSIVE (PASV)

This command requests the server-DTP to "listen" on a data port (which is not its default data port) and to wait for a connection rather than initiate one upon receipt of a transfer command. The response to this command includes the host and port address this server is listening on.

REPRESENTATION TYPE (TYPE)

The argument specifies the representation type as described in the Section on Data Representation and Storage. Several types take a second parameter. The first parameter is denoted by a single TELNET character, as is the second Format parameter for ASCII and EBCDIC; the second parameter for local byte is a decimal integer to indicate Bytesize. The parameters are separated by a <SP> (Space, ASCII code 32.).

The following codes are assigned for type:

A - ASCII	-> <-	N - Non-print
E - EBCDIC		T - TELNET format effectors
I - Image		C - Carriage Control (ASA)

L <byte size> - Local byte Byte size

The default representation type is ASCII Non-print. If the Format parameter is changed, and later just the first argument is changed, Format then returns to the Non-print default.

FILE STRUCTURE (STRU)

The argument is a single TELNET character code specifying file structure described in the Section on Data Representation and Storage.

The following codes are assigned for structure:

- F - File (no record structure)
- R - Record structure
- P - Page structure

The default structure is File.

TRANSFER MODE (MODE)

The argument is a single TELNET character code specifying the data transfer modes described in the Section on Transmission Modes.

The following codes are assigned for transfer modes:

- S - Stream
- B - Block
- C - Compressed

The default transfer mode is Stream.

FTP SERVICE COMMANDS

The FTP service commands define the file transfer or the file system function requested by the user. The argument of an FTP service command will normally be a pathname. The syntax of pathnames must conform to server site conventions (with standard defaults applicable), and the language conventions of the TELNET connection. The suggested default handling is to use the last specified device, directory or file name, or the standard default defined for local users. The commands may be in any order except that a "rename from" command must be followed by a "rename to" command and the restart command must be followed by the interrupted service command. The data, when transferred in response to FTP service commands, shall always be sent over the data connection, except for certain informative replies. The following commands specify FTP service requests:

RETRIEVE (RETR)

This command causes the server-DTP to transfer a copy of the file, specified in the pathname, to the server- or user-DTP at the other end of the data connection. The status and contents of the file at the server site shall be unaffected.

STORE (STOR)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data as a file at the server site. If the file specified in the pathname exists at the server site then its contents shall be replaced by the data being transferred. A new file is created at the server site if the file specified in the pathname does not already exist.

APPEND (with create) (APPE)

This command causes the server-DTP to accept the data transferred via the data connection and to store the data in a file at the server site. If the file specified in the pathname exists at the server site, then the data shall be appended to that file; otherwise the file specified in the pathname shall be created at the server site.

MAIL FILE (MLFL)

The intent of this command is to enable a user at the user site to mail data (in form of a file) to another user at the server site. It should be noted that the files to be mailed are transmitted via the data connection in ASCII or EBCDIC type. (It is the user's responsibility to ensure that the type is correct.) These files should be inserted into the destination user's mailbox by the server in accordance with serving Host mail conventions. The mail may be marked as sent from the particular user HOST and the user specified by the 'USER' command. The argument field may contain a Host system ident, or it may be empty. If the argument field is empty or blank (one or more spaces), then the mail is destined for a printer or other designated place for general delivery site mail.

MAIL (MAIL)

This command allows a user to send mail that is NOT in a file over the TELNET connection. The argument field may contain system ident, or it may be empty. The ident is defined as above for the MLFL command. After the 'MAIL' command is received, the server is to treat the following lines as text of the mail sent by the user. The mail text is to be terminated by a line containing only a single period, that is, the character sequence "CRLF.CRLF". It is suggested that a modest volume of mail service should be free; i.e., it may be entered before a USER command.

MAIL SEND TO TERMINAL (MSND)

This command is like the MAIL command, except that the data is displayed on the addressed user's terminal, if such access is currently allowed, otherwise an error is returned.

MAIL SEND TO TERMINAL OR MAILBOX (MSOM)

This command is like the MAIL command, except that the data is displayed on the addressed user's terminal, if such access is currently allowed, otherwise the data is placed in the user's mailbox.

MAIL SEND TO TERMINAL AND MAILBOX (MSAM)

This command is like the MAIL command, except that the data is displayed on the addressed user's terminal, if such access is currently allowed, and, in any case, the data is placed in the user's mailbox.

MAIL RECIPIENT SCHEME QUESTION (MRSQ)

This FTP command is used to select a scheme for the transmission of mail to several users at the same host. The schemes are to list the recipients first, or to send the mail first.

MAIL RECIPIENT (MRCP)

This command is used to identify the individual recipients of the mail in the transmission of mail for multiple users at one host.

ALLOCATE (ALLO)

This command may be required by some servers to reserve sufficient storage to accommodate the new file to be transferred. The argument shall be a decimal integer representing the number of bytes (using the logical byte size) of storage to be reserved for the file. For files sent with record or page structure a maximum record or page size (in logical bytes) might also be necessary; this is indicated by a decimal integer in a second argument field of the command. This second argument is optional, but when present should be separated from the first by the three TELNET characters <SP> R <SP>. This command shall be followed by a STORE or APPEND command. The ALLO command should be treated as a NOOP (no operation) by those servers which do not require that the maximum size of the file be declared beforehand, and those servers interested in only the maximum record or page size should accept a dummy value in the first argument and ignore it.

RESTART (REST)

The argument field represents the server marker at which file transfer is to be restarted. This command does not cause file transfer but "spaces" over the file to the specified data checkpoint. This command shall be immediately followed by the appropriate FTP service command which shall cause file transfer to resume.

RENAME FROM (RNFR)

This command specifies the file which is to be renamed. This command must be immediately followed by a "rename to" command specifying the new file pathname.

RENAME TO (RNT0)

This command specifies the new pathname of the file specified in the immediately preceding "rename from" command. Together the two commands cause a file to be renamed.

ABORT (ABOR)

This command tells the server to abort the previous FTP service command and any associated transfer of data. The

abort command may require "special action", as discussed in the Section on FTP Commands, to force recognition by the server. No action is to be taken if the previous command has been completed (including data transfer). The TELNET connection is not to be closed by the server, but the data connection must be closed.

There are two cases for the server upon receipt of this command: (1) the FTP service command was already completed, or (2) the FTP service command is still in progress.

In the first case, the server closes the data connection (if it is open) and responds with a 226 reply, indicating that the abort command was successfully processed.

In the second case, the server aborts the FTP service in progress and closes the data connection, returning a 426 reply to indicate that the service request terminated in abnormally. The server then sends a 226 reply, indicating that the abort command was successfully processed.

DELETE (DELE)

This command causes the file specified in the pathname to be deleted at the server site. If an extra level of protection is desired (such as the query, "DO you really wish to delete?"), it should be provided by the user-FTP process.

CHANGE WORKING DIRECTORY (CWD)

This command allows the user to work with a different directory or dataset for file storage or retrieval without altering his login or accounting information. Transfer parameters are similarly unchanged. The argument is a pathname specifying a directory or other system dependent file group designator.

LIST (LIST)

This command causes a list to be sent from the server to the passive DTP. If the pathname specifies a directory, the server should transfer a list of files in the specified directory. If the pathname specifies a file then the server should send current information on the file. A null argument implies the user's current working or default

directory. The data transfer is over the data connection in type ASCII or type EBCDIC. (The user must ensure that the TYPE is appropriately ASCII or EBCDIC).

NAME-LIST (NLST)

This command causes a directory listing to be sent from server to user site. The pathname should specify a directory or other system-specific file group descriptor; a null argument implies the current directory. The server will return a stream of names of files and no other information. The data will be transferred in ASCII or EBCDIC type over the data connection as valid pathname strings separated by <CRLF> or <NL>. (Again the user must ensure that the TYPE is correct.)

SITE PARAMETERS (SITE)

This command is used by the server to provide services specific to his system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol. The nature of these services and the specification of their syntax can be stated in a reply to the HELP SITE command.

STATUS (STAT)

This command shall cause a status response to be sent over the TELNET connection in the form of a reply. The command may be sent during a file transfer (along with the TELNET IP and Synch signals--see the Section on FTP Commands) in which case the server will respond with the status of the operation in progress, or it may be sent between file transfers. In the latter case the command may have an argument field. If the argument is a pathname, the command is analogous to the "list" command except that data shall be transferred over the TELNET connection. If a partial pathname is given, the server may respond with a list of file names or attributes associated with that specification. If no argument is given, the server should return general status information about the server FTP process. This should include current values of all transfer parameters and the status of connections.

HELP (HELP)

This command shall cause the server to send helpful information regarding its implementation status over the TELNET connection to the user. The command may take an argument (e.g., any command name) and return more specific information as a response. The reply is type 211 or 214. It is suggested that HELP be allowed before entering a USER command. The server may use this reply to specify site-dependent parameters, e.g., in response to HELP SITE.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the server send an OK reply.

The File Transfer Protocol follows the specifications of the TELNET protocol for all communications over the TELNET connection. Since, the language used for TELNET communication may be a negotiated option, all references in the next two sections will be to the "TELNET language" and the corresponding "TELNET end of line code". Currently one may take these to mean NVT-ASCII and <CRLF>. No other specifications of the TELNET protocol will be cited.

FTP commands are "TELNET strings" terminated by the "TELNET end of line code". The command codes themselves are alphabetic characters terminated by the character <SP> (Space) if parameters follow and TELNET-EOL otherwise. The command codes and the semantics of commands are described in this section; the detailed syntax of commands is specified in the Section on Commands, the reply sequences are discussed in the Section on Sequencing of Commands and Replies, and scenarios illustrating the use of commands are provided in the Section on Typical FTP Scenarios.

FTP commands may be partitioned as those specifying access-control identifiers, data transfer parameters, or FTP service requests. Certain commands (such as ABOR, STAT, QUIT) may be sent over the TELNET connection while a data transfer is in progress. Some servers may not be able to monitor the TELNET and data connections simultaneously, in which case some special action will be necessary to get the server's attention. The exact form of the "special action" is undefined; but the following ordered format is tentatively recommended:

1. User system inserts the TELNET "Interrupt Process" (IP) signal in the TELNET stream.
2. User system sends the TELNET "Synch" signal
3. User system inserts the command (e.g., ABOR) in the TELNET stream.
4. Server PI., after receiving "IP", scans the TELNET stream for EXACTLY ONE FTP command.

(For other servers this may not be necessary but the actions listed above should have no unusual effect.)

FTP REPLIES

Replies to File Transfer Protocol commands are devised to ensure the synchronization of requests and actions in the process of file transfer, and to guarantee that the user process always knows the state of the Server. Every command must generate at least one reply, although there may be more than one; in the latter case, the multiple replies must be easily distinguished. In addition, some commands occur in sequential groups, such as USER, PASS and ACCT, or RNFR and RNT0. The replies show the existence of an intermediate state if all preceding commands have been successful. A failure at any point in the sequence necessitates the repetition of the entire sequence from the beginning.

The details of the command-reply sequence are made explicit in a set of state diagrams below.

An FTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is intended for the human user. It is intended that the three digits contain enough encoded information that the user-process (the User-PI) will not need to examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be server-dependent, so there are likely to be varying texts for each reply code.

Formally, a reply is defined to contain the 3-digit code, followed by Space <SP>, followed by one line of text (where some maximum line length has been specified), and terminated by the TELNET end-of-line code. There will be cases, however, where the text is longer than a single line. In these cases the complete text must

be bracketed so the User-process knows when it may stop reading the reply (i.e. stop processing input on the TELNET connection) and go do other things. This requires a special format on the first line to indicate that more than one line is coming, and another on the last line to designate it as the last. At least one of these must contain the appropriate reply code to indicate the state of the transaction. To satisfy all factions it was decided that both the first and last line codes should be the same.

Thus the format for multi-line replies is that the first line will begin with the exact required reply code, followed immediately by a Hyphen, "-" (also known as Minus), followed by text. The last line will begin with the same code, followed immediately by Space <SP>, optionally some text, and the TELNET end-of-line code.

For example:

```
123-First line
Second line
  234 A line beginning with numbers
123 The last line
```

The user-process then simply needs to search for the second occurrence of the same reply code, followed by <SP> (Space), at the beginning of a line, and ignore all intermediary lines. If an intermediary line begins with a 3-digit number, the Server must pad the front to avoid confusion.

This scheme allows standard system routines to be used for reply information (such as for the STAT reply), with "artificial" first and last lines tacked on. In the rare cases where these routines are able to generate three digits and a Space at the beginning of any line, the beginning of each text line should be offset by some neutral text, like Space.

This scheme assumes that multi-line replies may not be nested. We have found that, in general, nesting of replies will not occur, except for random system messages (also called spontaneous replies) which may interrupt another reply. System messages (i.e. those not processed by the FTP server) will NOT carry reply codes and may occur anywhere in the command-reply sequence. They may be ignored by the User-process as they are only information for the human user.

The three digits of the reply each have a special significance. This is intended to allow a range of very simple to very sophisticated response by the user-process. The first digit denotes whether the response is good, bad or incomplete. (Referring to the state diagram) an unsophisticated user-process will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A user-process that wants to know approximately what kind of error occurred (e.g. file system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information (e.g. RNT0 command without a preceding RNFR.)

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The requested action is being initiated; expect another reply before proceeding with a new command. (The user-process sending another command before the completion reply would be in violation of protocol; but server-FTP processes should queue any commands that arrive while a preceding command is in progress.) This type of reply can be used to indicate that the command was accepted and the user-process may now pay attention to the data connections, for implementations where simultaneous monitoring is difficult.

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The user should send another command specifying this information. This reply is used in command sequence groups.

4yz Transition Negative Completion reply

The command was not accepted and the requested action did not take place, but the error condition is temporary and the action may be requested again. The user should

return to the beginning of the command sequence, if any. It is difficult to assign a meaning to "transient", particularly when two distinct sites (Server and User-processes) have to agree on the interpretation. Each reply in the 4yz category might have a slightly different time value, but the intent is that the user-process is encouraged to try again. A rule of thumb in determining if a reply fits into the 4yz or the 5yz (Permanent Negative) category is that replies are 4yz if the commands can be repeated without any change in command form or in properties of the User or Server (e.g. the command is spelled the same with the same arguments used; the user does not change his file access or user name; the server does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not take place. The User-process is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct his User-process to reinitiate the command sequence by direct action at some point in the future (e.g. after the spelling has been changed, or the user has altered his directory status.)

The following function groupings are encoded in the second digit:

- x0z Syntax - These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, unimplemented or superfluous commands.
- x1z Information - These are replies to requests for information, such as status or help.
- x2z Connections - Replies referring to the TELNET and data connections.
- x3z Authentication and accounting - Replies for the login process and accounting procedures.
- x4z Unspecified as yet

x5z File system - These replies indicate the status of the Server file system vis-a-vis the requested transfer or other file system action.

The third digit gives a finer gradation of meaning in each of the function categories, specified by the second digit. The list of replies below will illustrate this. Note that the text associated with each reply is recommended, rather than mandatory, and may even change according to the command with which it is associated. The reply codes, on the other hand, must strictly follow the specifications in the last section; that is, Server implementations should not invent new codes for situations that are only slightly different from the ones described here, but rather should adapt codes already defined.

A command such as TYPE or ALLO whose successful execution does not offer the user-process any new information will cause a 200 reply to be returned. If the command is not implemented by a particular Server-FTP process because it has no relevance to that computer system, for example ALLO at a TOPS20 site, a Positive Completion reply is still desired so that the simple User-process knows it can proceed with its course of action. A 202 reply is used in this case with, for example, the reply text: "No storage allocation necessary." If, on the other hand, the command requests a non-site-specific action and is unimplemented, the response is 502. A refinement of that is the 504 reply for a command that IS implemented, but that requests an unimplemented parameter.

Reply Codes by Function Groups

200 Command okay
500 Syntax error, command unrecognized
 [This may include errors such as command line too long.]
501 Syntax error in parameters or arguments
202 Command not implemented, superfluous at this site.
502 Command not implemented
503 Bad sequence of commands
504 Command not implemented for that parameter

110 Restart marker reply.

In this case the text is exact and not left to the particular implementation; it must read:

MARK yyyy = mmmm

where yyyy is User-process data stream marker, and mmmm server's equivalent marker. (note the spaces between markers and "=".)

- 119 Terminal not available, will try mailbox.
- 211 System status, or system help reply
- 212 Directory status
- 213 File status
- 214 Help message
(on how to use the server or the meaning of a particular non-standard command. This reply is useful only to the human user.)
- 215 <scheme> is the preferred scheme.

- 120 Service ready in nnn minutes
- 220 Service ready for new user
- 221 Service closing TELNET connection
(logged out if appropriate)
- 421 Service not available, closing TELNET connection.
This may be a reply to any command if the service knows it must shut down.]
- 125 Data connection already open; transfer starting
- 225 Data connection open; no transfer in progress
- 425 Can't open data connection
- 226 Closing data connection;
requested file action successful (for example, file transfer or file abort.)
- 426 Connection closed; transfer aborted.
- 227 Entering Passive Mode. h1,h2,h3,h4,p1,p2

- 230 User logged in, proceed
- 530 Not logged in
- 331 User name okay, need password
- 332 Need account for login
- 532 Need account for storing files

- 150 File status okay; about to open data connection.
- 151 User not local; Will forward to <user>@<host>.
- 152 User Unknown; Mail will be forwarded by the operator.
- 250 Requested file action okay, completed.
- 350 Requested file action pending further information
- 450 Requested file action not taken:
file unavailable (e.g. file busy)
- 550 Requested action not taken:

file unavailable (e.g. file not found, no access)
451 Requested action aborted: local error in processing
551 Requested action aborted: page type unknown
452 Requested action not taken:
insufficient storage space in system
552 Requested file action aborted:
exceeded storage allocation (for current directory or
dataset)
553 Requested action not taken:
file name not allowed
354 Start mail input; end with <CR><LF>.<CR><LF>

Numeric Order List of Reply Codes

110 Restart marker reply.
In this case the text is exact and not left to the
particular implementation; it must read:
MARK yyyy = mmmm
where yyyy is User-process data stream marker, and mmmm
server's equivalent marker. (note the spaces between
markers and "=".)
119 Terminal not available, will try mailbox.
120 Service ready in nnn minutes
125 Data connection already open; transfer starting
150 File status okay; about to open data connection.
151 User not local; Will forward to <user>@<host>.
152 User Unknown; Mail will be forwarded by the operator.
200 Command okay
202 Command not implemented, superfluous at this site.
211 System status, or system help reply
212 Directory status
213 File status
214 Help message
(on how to use the server or the meaning of a particular
non-standard command. This reply is useful only to the
human user.)
215 <scheme> is the preferred scheme.
220 Service ready for new user
221 Service closing TELNET connection
(logged out if appropriate)
225 Data connection open; no transfer in progress
226 Closing data connection;
requested file action successful (for example, file transfer
or file abort.)
227 Entering Passive Mode. h1,h2,h3,h4,p1,p2

230 User logged in, proceed
250 Requested file action okay, completed.
331 User name okay, need password
332 Need account for login
350 Requested file action pending further information
354 Start mail input; end with <CR><LF>.<CR><LF>
421 Service not available, closing TELNET connection.
This may be a reply to any command if the service knows it
must shut down.]
425 Can't open data connection
426 Connection closed; transfer aborted.
450 Requested file action not taken:
file unavailable (e.g. file busy)
451 Requested action aborted: local error in processing
452 Requested action not taken:
insufficient storage space in system
500 Syntax error, command unrecognized
[This may include errors such as command line too long.]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command not implemented for that parameter
530 Not logged in
532 Need account for storing files
550 Requested action not taken:
file unavailable (e.g. file not found, no access)
551 Requested action aborted: page type unknown
552 Requested file action aborted:
exceeded storage allocation (for current directory or
dataset)
553 Requested action not taken:
file name not allowed

DECLARATIVE SPECIFICATIONS

MINIMUM IMPLEMENTATION

In order to make FTP workable without needless error messages, the following minimum implementation is required for all servers:

TYPE - ASCII Non-print
MODE - Stream
STRUCTURE - File, Record
COMMANDS - USER, QUIT, PORT,
 TYPE, MODE, STRU,
 for the default values
 RETR, STOR,
 NOOP.

The default values for transfer parameters are:

TYPE - ASCII Non-print
MODE - Stream
STRU - File

All Hosts must accept the above as the standard defaults.

CONNECTIONS

The server protocol interpreter shall "listen" on Port L. The user or user protocol interpreter shall initiate the full-duplex TELNET connection. Server- and user- processes should follow the conventions of the TELNET protocol as specified in the ARPA Internet Protocol Handbook. Servers are under no obligation to provide for editing of command lines and may specify that it be done in the user Host. The TELNET connection shall be closed by the server at the user's request after all transfers and replies are completed.

The user-DTP must "listen" on the specified data port; this may be the default user port (U) or a port specified in the PORT command. The server shall initiate the data connection from his own default data port (L-1) using the specified user data port. The direction of the transfer and the port used will be determined by the FTP service command.

When data is to be transferred between two servers, A and B (refer to Figure 2), the user-PI, C, sets up TELNET connections with both server-PI's. One of the servers, say A, is then sent a PASV command telling him to "listen" on his data port rather than initiate a connection when he receives a transfer service command. When the user-PI receives an acknowledgment to the PASV command, which includes the identity of the host and port being listened on, the user-PI then sends A's port, a, to B in a PORT command; a reply is returned. The user-PI may then send the corresponding service commands to A and B. Server B initiates the connection and the transfer proceeds. The command-reply sequence is listed below where the messages are vertically synchronous but horizontally asynchronous:

User-PI - Server A

C->A : Connect
C->A : PASV
A->C : 227 Entering Passive Mode.

C->A : STOR
B->A : Connect to HOST-A, PORT-a

User-PI - Server B

C->B : Connect

C->B : PORT A1,A2,A3,A4,a1,a2
B->C : 200 Okay
C->B : RETR

The data connection shall be closed by the server under the conditions described in the Section on Establishing Data Connections. If the server wishes to close the connection after a transfer where it is not required, he should do so immediately after the file transfer is completed. He should not wait until after a new transfer command is received because the user-process will have already tested the data connection to see if it needs to do a "listen"; (recall that the user must "listen" on a closed data port BEFORE sending the transfer request). To prevent a race condition here, the server sends a reply (226) after closing the data connection (or if the connection is left open, a "file transfer completed" reply (250) and the user-PI should wait for one of these replies before issuing a new transfer command.

COMMANDS

The commands are TELNET character string transmitted over the TELNET connections as described in the Section on FTP Commands. The command functions and semantics are described in the Section on Access Control Commands, Transfer Parameter Commands, FTP Service Commands, and Miscellaneous Commands. The command syntax is specified here.

The commands begin with a command code followed by an argument field. The command codes are four or fewer alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus any of the following may represent the retrieve command:

RETR Retr retr ReTr rETr

This also applies to any symbols representing parameter values, such as A or a for ASCII TYPE. The command codes and the argument fields are separated by one or more spaces.

The argument field consists of a variable length character string ending with the character sequence <CRLF> (Carriage Return, Linefeed) for NVT-ASCII representation; for other negotiated languages a different end of line character might be used. It should be noted that the server is to take NO action until the end of line code is received.

The syntax is specified below in NVT-ASCII. All characters in the argument field are ASCII characters including any ASCII represented decimal integers. Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

The following are the FTP commands:

```
USER <SP> <username> <CRLF>
PASS <SP> <password> <CRLF>
ACCT <SP> <account information> <CRLF>
REIN <CRLF>
QUIT <CRLF>
PORT <SP> <Host-port> <CRLF>
PASV <CRLF>
TYPE <SP> <type code> <CRLF>
STRU <SP> <structure code> <CRLF>
MODE <SP> <mode code> <CRLF>
RETR <SP> <pathname> <CRLF>
STOR <SP> <pathname> <CRLF>
APPE <SP> <pathname> <CRLF>
MLFL [<SP> <ident>] <CRLF>
MAIL [<SP> <ident>] <CRLF>
MSND [<SP> <ident>] <CRLF>
MSOM [<SP> <ident>] <CRLF>
MSAM [<SP> <ident>] <CRLF>
MRSQ [<SP> <scheme>] <CRLF>
MRCP <SP> <ident> <CRLF>
ALLO <SP> <decimal integer>
    [<SP> R <SP> <decimal integer>] <CRLF>
REST <SP> <marker> <CRLF>
RNFR <SP> <pathname> <CRLF>
RNT0 <SP> <pathname> <CRLF>
ABOR <CRLF>
DELE <SP> <pathname> <CRLF>
CWD <SP> <pathname> <CRLF>
LIST [<SP> <pathname>] <CRLF>
NLST [<SP> <pathname>] <CRLF>
SITE <SP> <string> <CRLF>
STAT [<SP> <pathname>] <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>
```

The syntax of the above argument fields (using BNF notation where applicable) is:

```
<username> ::= <string>
<password> ::= <string>
<account information> ::= <string>
<string> ::= <char> | <char><string>
<char> ::= any of the 128 ASCII characters except <CR> and <LF>
<marker> ::= <pr string>
<pr string> ::= <pr char> | <pr char><pr string>
<pr char> ::= printable characters, any
               ASCII code 33 through 126
<byte size> ::= any decimal integer 1 through 255
<Host-port> ::= <Host-number>,<Port-number>
<Host-number> ::= <number>,<number>,<number>,<number>
<Port-number> ::= <number>,<number>
<number> ::= any decimal integer 0 through 255
<ident> ::= <string>
<scheme> ::= R | T | ?
<form code> ::= N | T | C
<type code> ::= A [<SP> <form code>]
               | E [<SP> <form code>]
               | I
               | L <SP> <byte size>
<structure code> ::= F | R | P
<mode code> ::= S | B | C
<pathname> ::= <string>
```

SEQUENCING OF COMMANDS AND REPLIES

The communication between the user and server is intended to be an alternating dialogue. As such, the user issues an FTP command and the server responds with a prompt primary reply. The user should wait for this initial primary success or failure response before sending further commands.

Certain commands require a second reply for which the user should also wait. These replies may, for example, report on the progress or completion of file transfer or the closing of the data connection. They are secondary replies to file transfer commands.

One important group of informational replies is the connection greetings. Under normal circumstances, a server will send a 220 reply, "awaiting input", when the connection is completed. The user should wait for this greeting message before sending any commands. If the server is unable to accept input right away, he should send a 120 "expected delay" reply immediately and a 220 reply when ready. The user will then know not to hang up if there is a delay.

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a server may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

Command-Reply Sequences

In this section, the command-reply sequence is presented. Each command is listed with its possible replies; command groups are listed together. Preliminary replies are listed first (with their succeeding replies indented and under them), then positive and negative completion, and finally intermediary replies with the remaining commands from the sequence following. This listing forms the basis for the state diagrams, which will be presented separately.

Connection Establishment

120
220
220
421

```
Login
  USER
    230
    530
    500, 501, 421
    331, 332
  PASS
    230
    202
    530
    500, 501, 503, 421
    332
  ACCT
    230
    202
    530
    500, 501, 503, 421
Logout
  QUIT
    221
    500
  REIN
    120
    220
    220
    421
    500, 502
Transfer parameters
  PORT
    200
    500, 501, 421, 530
  PASV
    227
    500, 501, 502, 421, 530
  MODE, TYPE, STRU
    200
    500, 501, 504, 421, 530
File action commands
  ALLO
    200
    202
    500, 501, 504, 421, 530
  REST
    500, 501, 502, 421, 530
    350
```

IEN 149
RFC 765

June 1980
File Transfer Protocol

STOR
125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 452, 553
500, 501, 421, 530
RETR
125, 150
(110)
226, 250
425, 426, 451
450, 550
500, 501, 421, 530
LIST, NLST
125, 150
226, 250
425, 426, 451
450
500, 501, 502, 421, 530
APPE
125, 150
(110)
226, 250
425, 426, 451, 551, 552
532, 450, 550, 452, 553
500, 501, 502, 421, 530
MLFL
125, 150, 151, 152
226, 250
425, 426, 451, 552
532, 450, 550, 452, 553
500, 501, 502, 421, 530
RNFR
450, 550
500, 501, 502, 421, 530
350
RNT0
250
532, 553
500, 501, 502, 503, 421, 530
DELE, CWD
250
450, 550
500, 501, 502, 421, 530

ABOR
225, 226
500, 501, 502, 421
MAIL, MSNC
151, 152
354
250
451, 552
354
250
451, 552
450, 550, 452, 553
500, 501, 502, 421, 530
MSOM, MSAM
119, 151, 152
354
250
451, 552
354
250
451, 552
450, 550, 452, 553
500, 501, 502, 421, 530
MRSQ
200, 215
500, 501, 502, 421, 530
MRCP
151, 152
200
200
450, 550, 452, 553
500, 501, 502, 503, 421
Informational commands
STAT
211, 212, 213
450
500, 501, 502, 421, 530
HELP
211, 214
500, 501, 502, 421
Miscellaneous commands
SITE
200
202
500, 501, 530

IEN 149
RFC 765

June 1980
File Transfer Protocol

NOOP
200
500 421

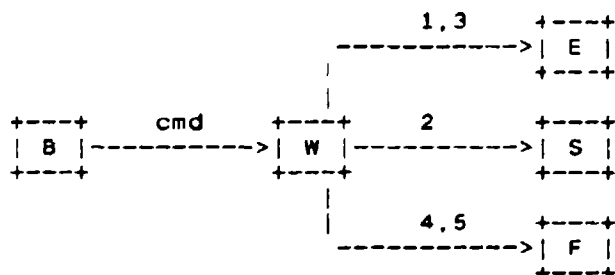
STATE DIAGRAMS

Here we present state diagrams for a very simple minded FTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of FTP commands or command sequences.

The command groupings were determined by constructing a model for each command then collecting together the commands with structurally identical models.

For each command or command sequence there are three possible outcomes: success (S), failure (F), and error (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

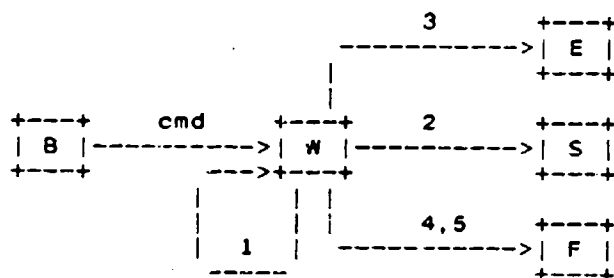
We first present the diagram that represents the largest group of FTP commands:



This diagram models the commands:

ABOR, ALLO, DELE, CWD, HELP, MODE, MRCP, MRSQ, NOOP, PASV,
QUIT, SITE, PORT, STAT, STRU, TYPE.

The other large group of commands is represented by a very similar diagram:

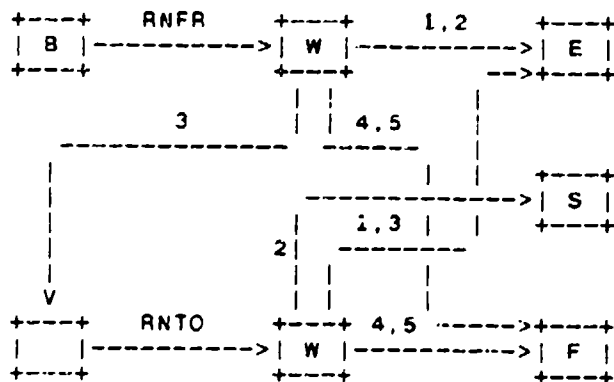


This diagram models the commands:

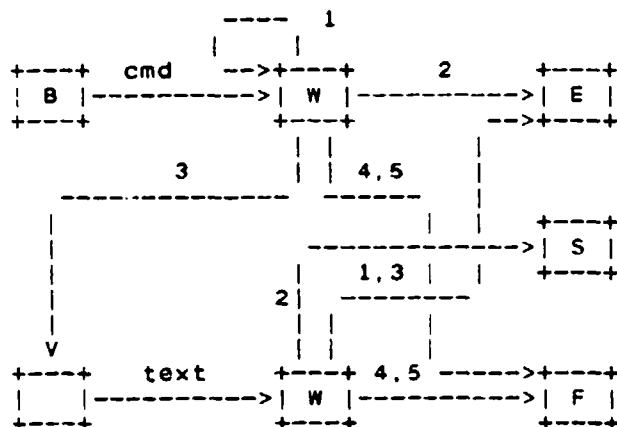
APPE, LIST, MLFL, NLST, REIN, RETR, STOR.

Note that this second model could also be used to represent the first group of commands, the only difference being that in the first group the 100 series replies are unexpected and therefore treated as error, while the second group expects (some may require) 100 series replies.

The remaining diagrams model command sequences, perhaps the simplest of these is the rename sequence:



A very similar diagram models the Mail and Send commands:



This diagram models the commands:

MAIL, MSND, MSOM, MSAM.

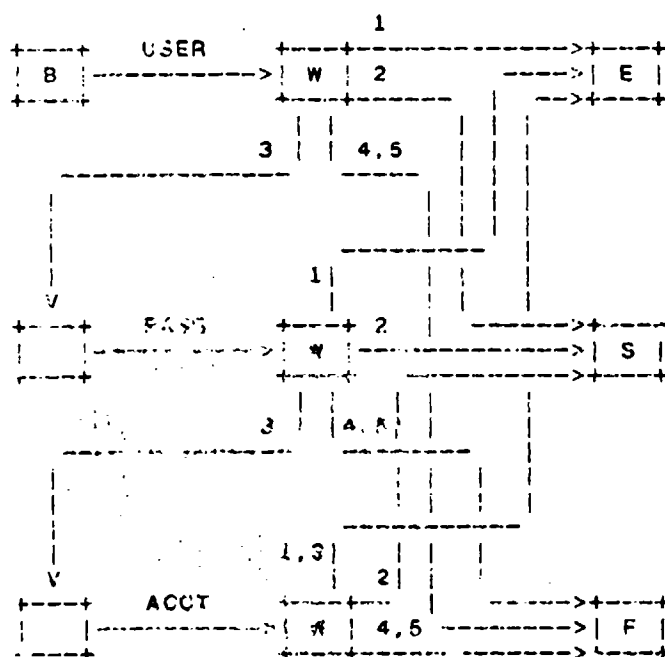
Note that the "text" here is a series of lines sent from the user to the server with no response expected until the last line is sent. recall that the last line must consist only of a single period.

```

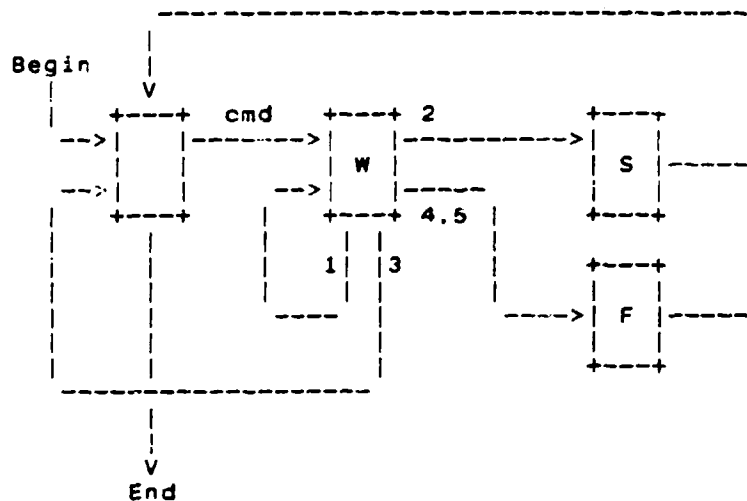
graph TD
    B[B] -- REST --> W[W]
    W -- "1, 2" --> E[E]
    E -- "3" --> S[S]
    E -- "4, 5" --> V[V]
    S -- "2" --> V
    S -- "3" --> F[F]
    V -- "cmd" --> W
    V -- "4, 5" --> F
    W -- "1" --> F
  
```

We note that the above three models are similar, in fact the Mail diagram and the Rename diagram are structurally identical. The Restart differs from the other two only in the treatment of 100 series replies at the second stage.

The most complicated diagram is for the Login sequence:



Finally we present a generalized diagram that could be used to model the command and reply interchange:



TYPICAL FTP SCENARIO

User at Host U wanting to transfer files to/from Host S:

In general the user will communicate to the server via a mediating user-FTP process. The following may be a typical scenario. The user-FTP prompts are shown in parentheses, '---->' represents commands from Host U to Host S, and '<----' represents replies from Host S to Host U.

LOCAL COMMANDS BY USER	ACTION INVOLVED
ftp (host) multics<CR>	Connect to Host S, port L, establishing TELNET connections <---- 220 Service ready <CRLF>
username Doe <CR>	USER Doe<CRLF>----> <---- 331 User name ok, need password<CRLF>
password mumble <CR>	PASS mumble<CRLF>----> <---- 230 User logged in.<CRLF>
retrieve (local type) ASCII<CR> (local pathname) test 1 <CR> (for.pathname) test.pl1<CR>	User-FTP opens local file in ASCII. RETR test.pl1<CRLF> ----> <---- 150 File status okay; about to open data connection Server makes data connection to port U
<CRLF>	<---- 226 Closing data connection, file transfer successful<CRLF>
type Image<CR>	TYPE I<CRLF> ----> <---- 200 Command OK<CRLF>
store (local type) image<CR> (local pathname) file dump<CR> (for.pathname) >udd>cn>fd<CR>	User-FTP opens local file in Image. STOR >udd>cn>fd<CRLF> ----> <---- 450 Access denied<CRLF>
terminate	QUIT <CRLF> ----> Server closes all connections.

IEN 149
RFC 765

June 1980
File Transfer Protocol

CONNECTION ESTABLISHMENT

The FTP control connection is established via TCP between the user process port U and the server process port L. This protocol is assigned the service port 21 (25 octal), that is L=21.

APPENDIX ON MAIL

The basic commands transmitting mail are the MAIL and the MLFL commands. These commands cause the transmitted data to be entered into the recipients mailbox.

MAIL <SP> <recipient name> <CRLF>

If accepted, returns 354 reply and considers all succeeding lines to be the message text, terminated by a line containing only a period, upon which a 250 completion reply is returned. Various errors are possible.

MLFL <SP> <recipient name> <CRLF>

If accepted, acts like a STOR command, except that the data is considered to be the message text. Various errors are possible.

There are two possible preliminary replies that a server may use to indicate that it is accepting mail for a user whose mailbox is not at that server.

151 User not local; Will forward to <user>@<host>.

This reply indicates that the server knows the user's mailbox is on another host and will take responsibility for forwarding the mail to that host. For example, at BBN (or ISI) there are several host which each have a list of many of the users on several of the host. These hosts then can accept mail for any user on their list and forward it to the correct host.

152 User Unknown; Mail will be forwarded by the operator.

This reply indicates that the host does not recognize the user name, but that it will accept the mail and have the operator attempt to deliver it. This is useful if the user name is misspelled, but may be a disservice if the mail is really undeliverable.

Three FTP commands provide for "sending" a message to a logged-in user's terminal, as well as variants for mailing it normally whether the user is logged in or not.

MSND -- SEND to terminal.

Returns 450 failure reply if the addressee is refusing or not logged in.

MSOM -- Send to terminal Or Mailbox.

Returns 119 notification reply if terminal is not accessible.

MSAM -- Send to terminal And Mailbox.

Returns 119 notification reply if terminal is not accessible.

Note that for MSOM and MSAM, it is the mailing which determines success, not the sending, although MSOM as implemented uses a 119 reply (in addition to the normal success/failure code) to indicate that because the SEND failed, an attempt is being made to mail the message instead. There are no corresponding variants for MLFL, since messages transmitted in this way are generally short.

There are two FTP commands which allow one to mail the text of a message to several recipients simultaneously; such message transmission is far more efficient than the practice of sending the text again and again for each additional recipient at a site.

There are two basic ways of sending a single text to several recipients. In one, all recipients are specified first, and then the text is sent; in the other, the order is reversed and the text is sent first, followed by the recipients. Both schemes are necessary because neither by itself is optimal for all systems, as will be explained later. To select a particular scheme, the MRSQ command is used; to specify recipients after a scheme is chosen, MRCP commands are given; and to furnish text, the MAIL or MLFL commands are used.

Scheme Selection: MRSQ

MRSQ is the means by which a user program can test for implementation of MRSQ/MRCP, select a particular scheme, reset its state thereof, and even do some rudimentary negotiation. Its format is like that of the TYPE command, as follows:

MRSQ [<SP> <scheme>] <CRLF>

<scheme> = a single character. The following are defined:

- R Recipients first. If not implemented, T must be.
- T Text first. If this is not implemented, R must be.
- ? Request for preference. Must always be implemented.

No argument means a "selection" of none of the schemes (the default).

Replies:

- 200 OK, we'll use specified scheme.
- 215 <scheme> This is the scheme I prefer.
- 501 I understand MRSQ but can't use that scheme.
- 5xx Command unrecognized or unimplemented.

Three aspects of MRSQ need to be pointed out here. The first is that an MRSQ with no argument must always return a 200 reply and restore the default state of having no scheme selected. Any other reply implies that MRSQ and hence MRCP are not understood or cannot be performed correctly.

The second is that the use of "?" as a <scheme> asks the FTP server to return a 215 reply in which the server specifies a "preferred" scheme. The format of this reply is simple:

215 <SP> <scheme> [<SP> <arbitrary text>] <CRLF>

Any other reply (e.g. 4xx or 5xx) implies that MRSQ and MRCP are not implemented, because "?" must always be implemented if MRSQ is.

The third important thing about MRSQ is that it always has the side effect of resetting all schemes to their initial state. This reset must be done no matter what the reply will be - 200, 215, or 501. The actions necessary for a reset will be explained when discussing how each scheme actually works.

Message Text Specification: MAIL/MLFL

Regardless of which scheme (if any) has been selected, a MAIL or MLFL with a non-null argument will behave exactly as before; the MRSQ/MRCP commands have no effect on them. However, such normal MAIL/MLFL commands do have the same side effect as MRSQ: they "reset" the current scheme to its initial state.

It is only when the argument is null (e.g. MAIL<CRLF> or MLFL<CRLF>) that the particular scheme being used is important, because rather than producing an error (as most servers currently do), the server will accept message text for this "null" specification; what it does with it depends on which scheme is in effect, and will be described in "Scheme Mechanics".

Recipient specification: MRCP

In order to specify recipient names (i.e., idents) and receive some acknowledgment (or refusal) for each name, the following command is used:

MRCP <SP> <ident> <CRLF>

Reply for no scheme:

503 No scheme specified yet; use MRSQ.

Replies for scheme T are identical to those for MAIL/MLFL.

Replies for scheme R (recipients first):

200 OK, name stored.

452 Recipient table full, this name not stored.

553 Recipient name rejected.

4xx Temporary error, try this name again later.

5xx Permanent error, report to sender.

Note that use of this command is an error if no scheme has been selected yet; an MRSQ <scheme> must have been given if MRCP is to be used.

Scheme mechanics: MRSQ R (Recipients first)

In the recipients-first scheme, MRCP is used to specify names which the FTP server stores in a list or table. Normally the reply for each MRCP will be either a 200 for acceptance, or a 4xx/5xx code for rejection; all 5xx codes are permanent rejections (e.g. user not known) which should be reported to the human sender, whereas 4xx codes in general connote some temporary error that may be rectified later. None of the 4xx/5xx replies impinge on previous or succeeding MRCP commands, except for 452 which indicates that no further MRCP's will succeed unless a message is sent to the already stored recipients or a reset is done.

Sending message text to stored recipients is done by giving a MAIL or MLFL command with no argument; that is, just MAIL<CRLF> or MLFL<CRLF>. Transmission of the message text is exactly the same as for normal MAIL/MLFL; however, a positive acknowledgment at the

end of transmission means that the message has been sent to ALL recipients that were remembered with MRCP, and a failure code means that it should be considered to have failed for ALL of these specified recipients. This applies regardless of the actual error code; and whether the reply signifies success or failure, all stored recipient names are flushed and forgotten - in other words, things are reset to their initial state. This purging of the recipient name list must also be done as the "reset" side effect of any use of MRSQ.

A 452 reply to an MRCP can thus be handled by using a MAIL/MLFL to specify the message for currently stored recipients, and then sending more MRCP's and another MAIL/MLFL, as many times as necessary; for example, if a server only had room for 10 names this would result in a 50-recipient message being sent 5 times, to 10 different recipients each time.

If a user attempts to specify message text (MAIL/MLFL with no argument) before any successful MRCP's have been given, this should be treated exactly as a "normal" MAIL/MLFL with a null recipient would be; some servers will return an error of some type, such as "550 Null recipient".

See Example 1 for an example using MRSQ R.

Scheme mechanics: MRSQ T (Text first)

In the text-first scheme, MAIL/MLFL with no argument is used to specify message text, which the server stores away. Succeeding MRCP's are then treated as if they were MAIL/MLFL commands, except that none of the text transfer manipulations are done; the stored message text is sent to the specified recipient, and a reply code is returned identical to that which an actual MAIL/MLFL would invoke. (Note ANY 2xx code indicates success.)

The stored message text is not forgotten until the next MAIL/MLFL or MRSQ, which will either replace it with new text or flush it entirely. Any use of MRSQ will reset this scheme by flushing stored text, as will any use of MAIL/MLFL with a non-null argument.

If an MRCP is seen before any message text has been stored, the user in effect is trying to send a null message; some servers might allow this, others would return an error code.

See Example 2 for an example using MRSQ T.

Why two schemes anyway?

Because neither by itself is optimal for all systems. MRSQ R allows more of a "bulk" mailing, because everything is saved up and then mailed simultaneously; this is very useful for systems such as ITS where the FTP server does not itself write mail directly, but hands it on to a central mailer demon of great power; the more information (e.g. recipients) associated with a single "hand-off", the more efficiently mail can be delivered.

By contrast, MRSQ T is geared to FTP servers which want to deliver mail directly in one-by-one incremental fashion. This way they can return an individual success/failure reply code for each recipient given which may depend on variable file system factors such as exceeding disk allocation, mailbox access conflicts, and so forth; if they tried to emulate MRSQ R's bulk mailing, they would have to ensure that a success reply to the MAIL/MLFL indeed meant that it had been delivered to ALL recipients specified - not just some.

Notes:

- Because these commands are not required in the minimum implementation of FTP, one must be prepared to deal with sites which don't recognize either MRSQ or MRCP. "MRSQ" and "MRSQ ?" are explicitly designed as tests to see whether either scheme is implemented; MRCP is not, and a failure return of the "unimplemented" variety could be confused with "No scheme selected yet", or even with "Recipient unknown". Be safe, be sure, use MRSQ!
- There is no way to indicate in a positive response to "MRSQ ?" that the preferred "scheme" for a server is that of the default state; i.e. none of the multi-recipient schemes. The rationale is that in this case, it would be pointless to implement MRSQ/MRCP at all, and the response would therefore be negative.
- One reason that the use of MAIL/MLFL is restricted to null arguments with this multi-recipient extension is the ambiguity that would result if a non-null argument were allowed; for example, if MRSQ R was in effect and some MRCP's had been given, and a MAIL FOO<CRLF> was done, there would be no way to distinguish a failure reply for mailbox "FOO" from a global failure for all recipients specified. A similar situation exists for MRSQ T; it would not be clear whether the text was stored and the mailbox failed, or vice versa, or both.

- "Resets" are done by all "MRSQ's and "normal" MAIL/MLFL's to avoid confusion and overly complicated implementation. The MRSQ command implies a change or uncertainty of status, and the latter commands would otherwise have to use some independent mechanisms to avoid clobbering the data bases (e.g., message text storage area) used by the T/R schemes. However, once a scheme is selected, it remains "in effect" just as a "TYPE A" remains selected. The recommended way for doing a reset, without changing the current selection, is with "MRSQ ?". Remember that "MRSQ" alone reverts to the no-scheme state.
- It is permissible to intersperse other FTP commands among the MRSQ/MRCP/MAIL sequences.

Example 1

Example of MRSQ R (Recipients first)

This is an example of how MRSQ R is used; first the user must establish that the server in fact implements MRSQ:

U: MRSQ
S: 200 OK, no scheme selected.

An MRSQ with a null argument always returns a 200 if implemented, selecting the "scheme" of null, i.e. none of them. If MRSQ were not implemented, a code of 4xx or 5xx would be returned.

U: MRSQ R
S: 200 OK, using that scheme

All's well; now the recipients can be specified.

U: MRCP Foo
S: 200 OK

U: MRCP Raboof
S: 553 Who's that? No such user here.

U: MRCP bar
S: 200 OK

Well, two out of three ain't bad. Note that the demise of "Raboof" has no effect on the storage of "Foo" or "bar". Now to furnish the message text, by giving a MAIL or MLFL with no argument:

U: MAIL
S: 354 Type mail, ended by <CRLF>.<CRLF>
U: Blah blah blah blah....etc etc etc
U: .
S: 250 Mail sent.

The text has now been sent to both "Foo" and "bar".

Example 2

Example of MRSQ T (Text first)

Using the same message as the previous example:

U: MRSQ ?
S: 215 T Text first, please.

MRSQ is indeed implemented, and the server says that it prefers "T", but that needn't stop the user from trying something else:

U: MRSQ R
S: 501 Sorry, I really can't do that.

It's possible that it could have understood "R" also, but in general it's best to use the "preferred" scheme, since the server knows which is most efficient for its particular site. Anyway:

U: MRSQ T
S: 200 OK, using that scheme.

Scheme "T" is now selected, and the text must be sent:

U: MAIL
S: 354 Type mail, ended by <CRLF>.<CRLF>
U: Blah blah blah blah....etc etc etc
U: .
S: 250 Mail stored.

Now recipients can be specified:

U: MRCP Foo
S: 250 Stored mail sent.

U: MRCP Raboof
S: 553 Who's that? No such user here.

U: MRCP bar
S: 250 Stored mail sent.

IEN 149
RFC 765

June 1980
File Transfer Protocol

Again, the text has now been sent to both "Foo" and "bar", and still remains stored. A new message can be sent with another MAIL/MRCP... sequence, but the fastidious or paranoid could chose to do:

U: MRSQ ?
S: 215 T Text first, please.

Which resets things without altering the scheme in effect.

APPENDIX ON PAGE STRUCTURE

The need for FTP to support page structure derives principally from the need to support efficient transmission of files between TOPS20 systems, particularly the files used by NLS.

The file system of TOPS20 is based on the concept of pages. The system level is most efficient at manipulating files as pages. System level programs provide an interface to the file system so that many applications view files as sequential streams of characters. However, a few applications use the underlying page structures directly, and some of these create holey files.

A TOPS20 file is just a bunch of words pointed to by a page table. If those words contain CRLF's, fine -- but that doesn't mean "record" to TOPS20.

A TOPS20 disk file consists of four things: a pathname, a page table, a (possibly empty) set of pages, and a set of attributes.

The pathname is specified in the RETR or STOR command. It includes the directory name, file name, file name extension, and version number.

The page table contains up to 2^{18} entries. Each entry may be EMPTY, or may point to a page. If it is not empty, there are also some page-specific access bits; not all pages of a file need have the same access protection.

A page is a contiguous set of 512 words of 36 bits each.

The attributes of the file, in the File Descriptor Block (FDB), contain such things as creation time, write time, read time, writer's byte-size, end of file pointer, count of reads and writes, backup system tape numbers, etc.

Note that there is NO requirement that pages in the page table be contiguous. There may be empty page table slots between occupied ones. Also, the end of file pointer is simply a number. There is no requirement that it in fact point at the "last" datum in the file. Ordinary sequential I/O calls in TOPS20 will cause the end of file pointer to be left after the last datum written, but other operations may cause it not to be so, if a particular programming system so requires.

In fact both of these special cases, "holey" files and end-of-file pointers not at the end of the file, occur with NLS data files.

The TOPS20 paged files can be sent with the FTP transfer parameters: TYPE L 36, STRU P, and MODE S (in fact any mode could be used).

Each page of information has a header. Each header field, which is a logical byte, is a TOPS20 word, since the TYPE is L 36.

The header fields are:

Word 0: Header Length.

The header length is 5.

Word 1: Page Index.

If the data is a disk file page, this is the number of that page in the file's page map. Empty pages (holes) in the file are simply not sent. Note that a hole is NOT the same as a page of zeros.

Word 2: Data Length.

The number of data words in this page, following the header. Thus the total length of the transmission unit is the Header Length plus the Data Length.

Word 3: Page Type.

A code for what type of chunk this is. A data page is type 3, the FDB page is type 2.

Word 4: Page Access Control.

The access bits associated with the page in the file's page map. (This full word quantity is put into AC2 of an SPACS by the program reading from net to disk.)

After the header are Data Length data words. Data Length is currently either 512 for a data page or 21 for an FDB. Trailing zeros in a disk file page may be discarded, making Data Length less than 512 in that case.

June 1980
File Transfer Protocol

IEN 149
RFC 765

Data transfers are implemented like the layers of an onion: some characters are packaged into a line. Some lines are packaged into a file. The file is broken into other manageable units for transmission. Those units have compression applied to them. The units may be flagged by restart markers. On the other end, the process is reversed.

RFC-788

SIMPLE MAIL TRANSFER PROTOCOL

November 1981

(313)

TABLE OF CONTENTS

1. INTRODUCTION	1
2. THE SMTP MODEL	2
3. THE SMTP PROCEDURE	4
3.1. Mail	4
3.2. Forwarding	7
3.3. Verifying and Expanding	8
3.4. Sending and Mailing	10
3.5. Opening and Closing	12
3.6. Relaying	13
3.7. Domains	15
4. THE SMTP SPECIFICATIONS	16
4.1. SMTP Commands	16
4.1.1. Command Semantics	16
4.1.2. Command Syntax	23
4.2. SMTP Replies	28
4.2.1. Reply Codes by Function Group	29
4.2.2. Reply Codes in Numeric Order	30
4.3. Sequencing of Commands and Replies	31
4.4. State Diagrams	33
4.5. Details	35
4.5.1. Minimum Implementation	35
4.5.2. Transparency	35
4.5.3. Sizes	36
APPENDIX A: TCP	38
APPENDIX B: NCP	39
APPENDIX C: NITS	40
APPENDIX D: X.25	41
APPENDIX E: Theory of Reply Codes	42
APPENDIX F: Scenarios	45
GLOSSARY	58
REFERENCES	61

SIMPLE MAIL TRANSFER PROTOCOL

1. INTRODUCTION

The objective of Simple Mail Transfer Protocol (SMTP) is to transfer mail reliably and efficiently.

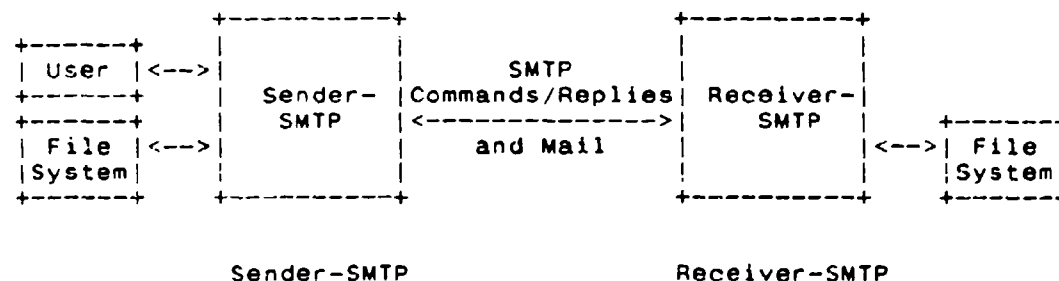
SMTP is independent of the particular transmission subsystem and requires only a reliable ordered data stream channel. Appendices A, B, C, and D describe the use of SMTP with various transport services. A Glossary provides the definitions of terms as used in this document.

An important feature of SMTP is its capability to relay mail across transport service environments. A transport service provides an interprocess communication environment (IPCE). An IPCE may cover one network, several networks, or a subset of a network. It is important to realize that transport systems (or IPCEs) are not one-to-one with networks. A process can communicate directly with another process through any mutually known IPCE. Mail is an application or use of interprocess communication. Mail can be communicated between processes in different IPCEs by relaying through a process connected to two (or more) IPCEs. More specifically, mail can be relayed between hosts on different transport systems by a host on both transport systems.

2. THE SMTP MODEL

The SMTP design is based on the following model of communication: as the result of a user mail request, the sender-SMTP establishes a full-duplex transmission channel to a receiver-SMTP. The receiver-SMTP may be either the ultimate destination or an intermediate. SMTP commands are generated by the sender-SMTP and sent to the receiver-SMTP. SMTP replies are sent from the receiver-SMTP to the sender-SMTP in response to the commands.

Once the transmission channel is established, the SMTP-sender sends a MAIL command indicating the sender of the mail. If the SMTP-receiver can accept mail it responds with an OK reply. The SMTP-sender then sends a RCPT command identifying a recipient of the mail. If the SMTP-receiver can accept mail for that recipient it responds with an OK reply; if not, it responds with a reply rejecting that recipient (but not the whole mail transaction). The SMTP-sender and SMTP-receiver may negotiate several recipients. When the recipients have been negotiated the SMTP-sender sends the mail data, terminating with a special sequence. If the SMTP-receiver successfully processes the mail data it responds with an OK reply. The dialog is purposely lock-step, one-at-a-time.



Model for SMTP Use

Figure 1

The SMTP provides mechanisms for the transmission of mail; directly from the sending user's host to the receiving user's host when the

two host are connected to the same transport service, or via one or more relay SMTP-servers when the source and destination hosts are not connected to the same transport service.

To be able to provide the relay capability the SMTP-server must be supplied with the name of the ultimate destination host as well as the destination mailbox name.

The argument to the MAIL command is a reverse-path, which specifies who the mail is from. The argument to the RCPT command is a forward-path, which specifies who the mail is to. The forward-path is a source route while the reverse-path, is a return route (which may be used to return a message to the sender when an error occurs with a relayed message).

When the same message is sent to multiple recipients the SMTP encourages the transmission of only one copy of the data for all the recipients at the same destination host.

The mail commands and replies have a rigid syntax. Replies also have a numeric code. In the following, examples appear which use actual commands and replies. The complete lists of commands and replies appears in Section 4 on specifications.

Commands and replies are not case sensitive. That is, a command or reply word may be upper case, lower case, or any mixture of upper and lower case. Note that this is not true of mailbox user names. For some hosts the user name is case sensitive, and SMTP implementations must take case to preserve the case of user names as they appear in mailbox arguments. Host names are not case sensitive.

Commands and replies are composed of characters from the ASCII character set [1]. Each 7-bit character is transmitted right justified in an 8-bit byte (or octet) with the high order bit cleared to zero.

When specifying the general form of a command or reply, an argument (or special symbol) will be denoted by a meta-linguistic variable (or constant), for example, "<string>" or "<reverse-path>". Here the angle brackets indicate these are a meta-linguistic variables. However, some arguments use the angle brackets literally. For example, an actual reverse-path is enclosed in angle brackets, i.e., "<Smith@ISIA>" is an instance of <reverse-path> (the angle brackets are actually transmitted in the command or reply).

3. THE SMTP PROCEDURES

This section presents the procedures used in SMTP in several parts. First comes the basic mail procedure defined as a mail transaction. Following this are descriptions of forwarding mail, verifying mailbox names and expanding mailing lists, sending to terminals instead of or in combination with mailboxes, and the opening and closing exchanges. At the end of this section are comments on relaying, and a note on mail domains. Throughout this section are examples of partial command and reply sequences, several complete scenarios are presented in Appendix F.

3.1. MAIL

There are three steps to a SMTP mail transaction. The transaction is started with a MAIL command which gives the sender identification. A series of one or more RCPT commands follow giving the receiver information. Then a DATA command gives the mail data. And finally, the end of mail data indicator confirms the transaction.

The first step in the procedure is the MAIL command. The <reverse-path> contains the source mailbox.

```
MAIL <SP> FROM:<reverse-path> <CRLF>
```

This command tells the the SMTP-receiver that a new mail transaction is starting and to reset all its state tables and buffers including any recipients or mail data. It gives the reverse-path which can be used to report errors. If accepted, the receiver-SMTP returns a 250 OK reply.

The <reverse-path> can contain more than just a mailbox. The <reverse-path> is a reverse source routing list of hosts and source mailbox. The first host in the <reverse-path> should be the host sending this command.

The second step in the procedure is the RCPT command.

```
RCPT <SP> TO:<forward-path> <CRLF>
```

This command gives a forward-path identifying one recipient. If accepted, the receiver-SMTP returns a 250 OK reply, and stores the forward-path. If the recipient is unknown the receiver-SMTP returns a 550 Failure reply. This second step of the procedure can be repeated any number of times.

The <forward-path> can contain more than just a mailbox. The <forward-path> is a source routing list of hosts and destination mailbox. The first host in the <forward-path> should be the host receiving this command.

The third step in the procedure is the DATA command.

DATA <CRLF>

If accepted, the receiver-SMTP returns a 354 Intermediate reply and considers all succeeding lines to be the message text. When the end of text is received and stored the SMTP-receiver sends a 250 OK reply.

Since the mail data is sent on the transmission channel the end of the mail data must be indicated so that the command and reply dialog can be resumed. SMTP indicates the end of the mail data by sending a line containing only a period. A transparency procedure is used to prevent this interfering with the user's text (see Section 4.5.2).

Please note that the mail data includes the memo header items such as Date, Subject, To, Cc, From [2].

The end of mail data indicator also confirms the mail transaction and tells the receiver-SMTP to now process the stored recipients and mail data. If accepted, the receiver-SMTP returns a 250 OK reply. The DATA command should fail only if the mail transaction was incomplete (for example, no recipients), or if resources are not available.

The above procedure is an example of a SMTP mail transaction. These commands must be used only in the order discussed above. Example 1 (below) illustrates the use of these commands in a mail transaction.

Example of the SMTP Procedure

This SMTP example shows mail sent by Smith at host Alpha, to Jones, Green, and Brown at host Beta. Here we assume that host Alpha contacts host Beta directly.

```
S: MAIL FROM:<Smith@Alpha>
R: 250 OK

S: RCPT TO:<Jones@Beta>
R: 250 OK

S: RCPT TO:<Green@Beta>
R: 550 No such user here

S: RCPT TO:<Brown@Beta>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: <CRLF>.<CRLF>
R: 250 OK
```

The mail has now been accepted for Jones and Brown. Green did not have a mailbox at host Beta.

Example 1

3.2. FORWARDING

There are some cases where the destination information in the <forward-path> is incorrect, but the receiver-SMTP knows the correct destination. In such cases, one of the following replies should be used to allow the sender to contact the correct destination.

251 User not local; will forward to <forward-path>

This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use in the future. Note that either the host or user or both may be different. The receiver takes responsibility for delivering the message.

551 User not local; please try <forward-path>

This reply indicates that the receiver-SMTP knows the user's mailbox is on another host and indicates the correct forward-path to use. Note that either the host or user or both may be different. The receiver refuses to accept mail for this user, and the sender must either redirect the mail according to the information provided or return an error response to the originating user.

Example 2 illustrates the use of these responses.

Example of Forwarding

Either

S: RCPT TO:<Postel@ISI>
R: 251 User not local; will forward to <Postel@ISIF>

Or

S: RCPT TO:<Paul@ISIB>
R: 551 User not local; please try <Mockapetris@ISIF>

Example 2

3.3. VERIFYING AND EXPANDING

SMTP provides as additional features, commands to verify a user name or expand a mailing list. This is done with the VRFY and EXPN commands, which have a character string arguments. For the VRFY command, the string is a user name, and the the response may include the full name of the user and must include the mailbox of the user. For the EXPN command, the string identifies a mailing list, and the multiline response may include the full name of the users and must give the mailboxes on the mailing list.

The case of verifying a user name is straightforward as shown in example 3.

Example of Verifying a User Name

Either

S: VRFY Postel
R: 250 Jon Postel <Postel@ISI>

Or

S: VRFY Jones
R: 550 String does not match anything.

Example 3

The case of expanding a mailbox list requires a multiline reply as shown in example 4.

Example of Expanding a Mailing List

Either

```
S: EXPN Example-People
R: 250-Jon Postel <Postel@ISIIF>
R: 250-Fred Fonebone <Fonebone@ISIQ>
R: 250-Sam Q. Smith <SQSmith@ISIQ>
R: 250-Quincy Smith <@ISIIF,Q-Smith@ISI-VAXA>
R: 250-<joe@foo-unix>
R: 250 <xyz@bar-unix>
```

Or

```
S: EXPN Executive-Washroom-List
R: 550 Access Denied to You.
```

Example 4

The character string arguments of the VRFY and EXPN commands cannot be further restricted due to the variety of implementations of the user name and mailbox list concepts. On some systems it may be appropriate for the argument of the EXPN command to be a file name for a file containing a mailing list, but again there is a variety of file naming conventions in the internet.

The VRFY and EXPN commands are not included in the minimum implementation (Section 4.5.1), and are not required to work across relays when they are implemented.

3.4. SENDING AND MAILING

The main purpose of SMTP is to deliver messages to user's mailboxes. A very similar service provided by some hosts is to deliver messages to user's terminals (provided the user is active on the host). The delivery to the user's mailbox is called "mailing", the delivery to the user's terminal is called "sending". Because in many hosts the implementation of sending is nearly identical to the implementation of mailing these two functions are combined in SMTP. However the sending commands are not included in the required minimum implementation (Section 4.5.1). User's should have the ability to control the writing of messages on their terminals. Most hosts permit the user's to accept or refuse such messages.

The following three command are defined to support the sending options, these are used in the mail transaction instead of the MAIL command and inform the receiver-SMTP of the special semantics of this transaction:

SEND <SP> FROM:<reverse-path> <CRLF>

The SEND command requires that the mail data be delivered to the user's terminal. If the user is not active (or not accepting terminal messages) on the host a 450 reply may be returned to a RCPT command. The mail transaction is successful if the message is delivered to the terminal.

SOML <SP> FROM:<reverse-path> <CRLF>

The Send Or Mail command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. If the user is not active (or not accepting terminal messages) then the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered either to the terminal or the mailbox.

SAML <SP> FROM:<reverse-path> <CRLF>

The Send And Mail command requires that the mail data be delivered to the user's terminal if the user is active (and accepting terminal messages) on the host. In any case the mail data is entered into the user's mailbox. The mail transaction is successful if the message is delivered to the mailbox.

The same reply codes that are used for the MAIL commands are used for these commands.

3.5. OPENING AND CLOSING

At the time the transmission channel is opened there is an exchange to ensure that the hosts are communicating with the hosts they think they are.

The following two commands are used in transmission channel opening and closing:

HELO <SP> <host> <CRLF>

QUIT <CRLF>

In the HELO command the host sending the command identifies itself; the command may be interpreted as saying "Hello, I am <host>".

Example of Connection Opening

R: 220 BBN-UNIX Simple Mail Transfer Service Ready
S: HELO USC-ISIF
R: 250 BBN-UNIX

Example 5

Example of Connection Closing

S: QUIT
R: 221 BBN-UNIX Service closing transmission channel

Example 6

3.6. RELAYING

The forward-path may be a source route of the form "@ONE,@TWO,JOE@THREE", where ONE, TWO, and THREE are hosts. This form is used to emphasize the distinction between an address and a route. The mailbox is an absolute address, and the route is information about how to get there. The two concepts should not be confused.

The elements of the forward-path are moved to the reverse-path as the message is relayed from one server-SMTP to another. The reverse-path is a reverse source route, (i.e., a source route from the current location of the message to the originator of the message). When a server-SMTP deletes its identifier from the forward-path and inserts it into the reverse-path, it must use the name it is known by in the environment it is sending into, not the environment the mail came from, in case the server-SMTP is known by different names in different environments.

Using source routing the receiver-SMTP receives mail to be relayed to another server-SMTP. The receiver-SMTP may accept or reject the task of relaying the mail in the same way it accepts or rejects mail for a local user. The receiver-SMTP transforms the command arguments by moving its own identifier from the forward-path to the beginning of the reverse-path. The receiver-SMTP then becomes a sender-SMTP, establishes a transmission channel to the next SMTP in the forward-path, and sends it the mail.

The first host in the reverse-path should be the host sending the SMTP commands, and the first host in the forward-path should be the host receiving the SMTP commands.

Notice that the forward-path and reverse-path appear in the SMTP commands and replies, but not necessarily in the message. That is, there is no need for these paths and especially this syntax to appear in the "To:", "From:", "CC:", etc. fields of the message header.

If a server-SMTP has accepted the task of relaying the mail and later finds that the forward-path is incorrect or that the mail cannot be delivered for whatever reason, then it must construct an "undeliverable mail" notification message and send it to the originator of the undeliverable mail (as indicated by the reverse-path).

This notification message must be from the server-SMTP at this host. Of course, server-SMTPs should not send notification messages about problems with notification messages. One way to prevent loops in error reporting is to specify a null reverse-path in the MAIL command of a notification message. When such a message is relayed it is permissible to leave the reverse-path null. A MAIL command with a null reverse-path appears as follows:

MAIL FROM:<>

An undeliverable mail notification message is shown in example 7. This notification is in response to a message originated by JOE at HOSTW and sent via HOSTX to HOSTY with instructions to relay it on to HOSTZ. What we see in the example is the transaction between HOSTY and HOSTX, which is the first step in the return of the notification message.

Example Undeliverable Mail Notification Message

```
S: MAIL FROM:<>
R: 250 ok
S: RCPT TO:<@HOSTX,JOE@HOSTW>
R: 250 ok
S: DATA
R: 354 send the mail data, end with .
S: Date: 23 Oct 81
S: Sender: SMTP@HOSTY
S: Subject: Mail System Problem
S:
S:   Sorry JOE, your message to SAM@HOSTZ lost.
S:   HOSTZ said this:
S:   "550 No Such User"
S: .
R: 250 ok
```

Example 7

3.7. DOMAINS

At some not too distant future time it might be necessary to expand the mailbox format to include a region or name domain identifier. There is quite a bit of discussion on this at present, and is likely that SMTP will be revised in the future to take into account naming domains.

The examples in this document do not show mail domains.

4. THE SMTP SPECIFICATIONS

4.1. SMTP COMMANDS

4.1.1. COMMAND SEMANTICS

The SMTP commands define the mail transfer or the mail system function requested by the user. SMTP commands are character strings terminated by <CRLF>. The command codes themselves are alphabetic characters terminated by <SP> if parameters follow and <CRLF> otherwise. The syntax of mailboxes must conform to receiver site conventions. The SMTP commands are discussed below. The SMTP replies are discussed in the Section 4.2.

A mail transaction involves several data objects which are communicated as arguments to different commands. The reverse-path is the argument of the MAIL command, the forward-path is the argument of the RCPT command, and the mail data is the argument of the DATA command. These arguments or data objects must be transmitted and held pending the confirmation communicated by the end of mail data indication which finalizes the transaction. The model for this is that distinct buffers are provided to hold the types of data objects, that is, there is a reverse-path buffer, a forward-path buffer, and a mail data buffer. Specific commands cause information to be appended to a specific buffer, or cause one or more buffers to be cleared.

HELLO (HELO)

This command is used to identify the sender-SMTP to the receiver-SMTP. The argument field contains the host name of the sender-SMTP.

The receiver-SMTP identifies itself to the sender-SMTP in the connection greeting reply, and in the response to this command.

MAIL (MAIL)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more mailboxes. The argument field contains a reverse-path

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it

is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different). In some types of error reporting messages (for example, undeliverable mail notifications) the reverse-path may be null (see Example 7).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

RECIPIENT (RCPT)

This command is used to identify an individual recipient of the mail data; multiple recipients are specified by multiple use of this command.

The forward-path consists of an optional list of hosts and a required destination mailbox. When the list of hosts is present, it is a source route and indicates that the mail must be relayed to the next host on the list. If the receiver-SMTP does not implement the relay function it may user the same reply it would for an unknown local user (550).

When mail is relayed, the relay host must remove itself from the beginning forward-path and put itself at the beginning of the reverse-path. When mail reaches its ultimate destination (the forward-path contains only a destination mailbox), the receiver-SMTP inserts it into the destination mailbox in accordance with its host mail conventions.

For example, mail received at relay host A with arguments

```
FROM:<X@Y>
TO:<@A,@B,C@D>
```

will be relayed on to host B with arguments

```
FROM:<@A,X@Y>
TO:<@B,C@D>.
```

This command causes its forward-path argument to be appended to the forward-path buffer.

DATA (DATA)

The receiver treats the lines following the command as mail data from the sender. This command causes the mail data from this command to be appended to the mail data buffer. The mail data may contain any of the 128 ASCII character codes.

The mail data is terminated by a line containing only a period, that is the character sequence "<CRLF>.<CRLF>" (see Section 4.5.2 on Transparency). This is the end of mail data indication.

The end of mail data indication requires that the receiver must now process the stored mail transaction information. This processing consumes the information in the reverse-path buffer, the forward-path buffer, and the mail data buffer, and on the completion of this command these buffers are cleared. If the processing is successful the receiver must send an OK reply. If the processing fails completely the receiver must send a failure reply.

When the receiver-SMTP accepts a message either for relaying or for final delivery it inserts at the beginning of the mail data a time stamp line. The time stamp line indicates the identity of the host that sent the message, and the identity of the host that received the message (and is inserting this time stamp), and the date and time the message was received. Relayed messages will have multiple time stamp lines.

When the receiver-SMTP makes the "final delivery" of a message it inserts at the beginning of the mail data a return path line. The return path line preserves the information in the <reverse-path> from the MAIL command. Here, final delivery means the message leaves the SMTP world. Normally, this would mean it has been delivered to the destination user, but in some cases it may be further processed and transmitted by another mail system.

The preceding two paragraphs imply that the final mail data

will begin with a return path line, followed by one or more time stamp lines. These lines will be followed by the mail data header and body [2]. For example:

```
Return-Path: <@GHI,@DEF,@ABC,JOE@ABC>
Mail-From: GHI received by JKL at 27-Oct-81 15:27:39-PST
Mail-From: DEF received by GHI at 27-Oct-81 15:15:13-PST
Mail-From: ABC received by DEF at 27-Oct-81 15:01:59-PST
Date: 27-Oct-81 15:01:01-PST
From: JOE@ABC
Subject: Improved Mailing System Installed
To: SAM@JKL
```

This is to inform you that ...

Special mention is needed of the response and further action required when the processing following the end of mail data indication is partially successful. This could arise if after accepting several recipients and the mail data, the receiver-SMTP finds that the mail data can be successfully delivered to some of the recipients, but it cannot be to others (for example, due to mailbox space allocation problems). In such a situation, the response to the DATA command must be an OK reply. But, the receiver-SMTP must compose and send an "undeliverable mail" notification message to the originator of the message. Either a single notification which lists all of the recipients that failed to get the message, or separate notification messages must be sent for each failed recipient (see Example 7). All undeliverable mail notification messages are sent using the MAIL command (even if they result from processing a SEND, SOML, or SAML command).

SEND (SEND)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals. The argument field contains a reverse-path. This command is successful if the message is delivered to the terminal.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender.

As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

SEND OR MAIL (SOML)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals or mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), otherwise to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to the terminal or the mailbox.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

SEND AND MAIL (SAML)

This command is used to initiate a mail transaction in which the mail data is delivered to one or more terminals and mailboxes. For each recipient the mail data is delivered to the recipient's terminal if the recipient is active on the host (and accepting terminal messages), and for all

recipients to the recipient's mailbox. The argument field contains a reverse-path. This command is successful if the message is delivered to the mailbox.

The reverse-path consists of an optional list of hosts and the sender mailbox. When the list of hosts is present, it is a "reverse" source route and indicates that the mail was relayed through each host on the list (the first host in the list was the most recent relay). This list is used as a source route to return non-delivery notices to the sender. As each relay host adds itself to the beginning of the list, it must use its name as known in the IPCE to which it is relaying the mail rather than the IPCE from which the mail came (if they are different).

This command clears the reverse-path buffer, the forward-path buffer, and the mail data buffer; and inserts the reverse-path information from this command into the reverse-path buffer.

RESET (RSET)

This command specifies that the current mail transaction is to be aborted. Any stored sender, recipients, and mail data must be discarded, and all buffers and state tables cleared. The receiver must send an OK reply.

VERIFY (VRFY)

This command asks the receiver to confirm that the argument identifies a user. If it is a user name, the full name of the user (if known) and the fully specified mailbox are returned.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

EXPAND (EXPN)

This command asks the receiver to confirm that the argument identifies a mailing list, and if so, to return the membership of that list. The full name of the users (if known) and the fully specified mailboxes are returned in a multiline reply.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

HELP (HELP)

This command causes the receiver to send helpful information to the sender of the HELP command. The command may take an argument (e.g., any command name) and return more specific information as a response.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

NOOP (NOOP)

This command does not affect any parameters or previously entered commands. It specifies no action other than that the receiver send an OK reply.

This command has no effect on any of the reverse-path buffer, the forward-path buffer, or the mail data buffer.

QUIT (QUIT)

This command specifies that the receiver must send an OK reply, and then close the transmission channel.

The receiver should not close the transmission channel until it receives and replies to a QUIT command (even if there was an error). The sender should not close the transmission channel until it send a QUIT command and receives the reply (even if there was an error response to a previous command). If the connection is closed prematurely the receiver should act as if a RSET command had been received (canceling any pending transaction, but not undoing any previously completed transaction), the sender should act as if the command or transaction in progress had received a temporary error (4xx).

There are restrictions on the order in which these command may be used.

The first command in a session must be the HELO command. The HELO command may be used later in a session as well.

The NOOP, HELP, EXPN, and VRFY commands can be used at any time during a session.

The MAIL, SEND, SOML, or SAML commands begin a mail transaction. Once started a mail transaction consists of one of the transaction beginning commands, one or more RCPT commands, and a DATA command, in that order. A mail transaction may be aborted by the RSET command. There may be zero or more transactions in a session.

The last command in a session must be the QUIT command. The QUIT command can not be used at any other time in a session.

4.1.2. COMMAND SYNTAX

The commands consist of a command code followed by an argument field. Command codes are four alphabetic characters. Upper and lower case alphabetic characters are to be treated identically. Thus, any of the following may represent the mail command:

MAIL Mail mail Mail mAI1

This also applies to any symbols representing parameter values, such as "TO" or "to" for the forward-path. Command codes and the argument fields are separated by one or more spaces. However, within the reverse-path and forward-path arguments case is important. In particular, in some hosts the user "smith" is different from the user "Smith".

The argument field consists of a variable length character string ending with the character sequence <CRLF>. The receiver is to take no action until this sequence is received.

Square brackets denote an optional argument field. If the option is not taken, the appropriate default is implied.

The following are the SMTP commands:

```
HELO <SP> <host> <CRLF>
MAIL <SP> FROM:<reverse-path> <CRLF>
RCPT <SP> TO:<forward-path> <CRLF>
DATA <CRLF>
RSET <CRLF>
SEND <SP> FROM:<reverse-path> <CRLF>
SOML <SP> FROM:<reverse-path> <CRLF>
SAML <SP> FROM:<reverse-path> <CRLF>
VRFY <SP> <string> <CRLF>
EXPN <SP> <string> <CRLF>
HELP [<SP> <string>] <CRLF>
NOOP <CRLF>
QUIT <CRLF>
```


The syntax of the above argument fields (using BNF notation where applicable) is given below. The "... " notation indicates that a field may be repeated one or more times.

```

<reverse-path> ::= <path>
<forward-path> ::= <path>
<path> ::= "<" ["@" <host> "." ...] <mailbox> ">"
<host> ::= <a> <string> | "#" <number> | "[" <dotnum> "]"
<mailbox> ::= <user> "@" <host>
<user> ::= <string>
<string> ::= <char> | <char> <string>
<char> ::= <c> | '\\' <c> | '\\' <s>
<dotnum> ::= <snum> "." <snum> "." <snum> "." <snum>
<number> ::= <d> | <d> <number>
<snum> ::= three digits representing a decimal integer value
           in the range 0 through 255
<a> ::= any one of the 52 alphabetic characters A through Z
        in upper case and a through z in lower case
<c> ::= any one of the 128 ASCII characters except
        <specials>
<d> ::= any one of the ten digits 0 through 9
<s> ::= any one of <specials>
<specials> ::= '<', '>', '(', ')', '\\', '\'', ';', ':', '@',
               '\'', and the control characters (ASCII codes 0 through 37
               octal inclusive and 177 octal)

```

Note that the backslash, '\\', is a quote character, which is used to indicate that the next character is to be used literally (instead of its normal interpretation). For example, "Joe\\.Smith" could be used to indicate a single nine character user field with comma being the fourth character of the field.

Hosts are generally known by names which are translated to addresses in each host. Sometimes a host is not known to the translation function and communication is blocked. To bypass this barrier two numeric forms are also allowed for host "names". One form is a decimal integer prefixed by a pound sign, "#", which indicates the number is the address of the host. Another form is four small decimal integers separated by dots and enclosed by brackets, e.g., "[123.255.37.2]", which indicates a 32-bit ARPA Internet Address in four 8-bit fields.

The time stamp line and the return path line are formally defined as follows:

<return-path-line> ::= "Return-Path:" <SP><reverse-path><CRLF>

<time-stamp-line> ::= "Mail-From:" <SP> <stamp> <CRLF>

<stamp> ::= [<ptcl>] <from-host> <this-host> <daytime>

<ptcl> ::= <protocol> <SP> "host" <SP>

<from-host> ::= <host> <SP>

<this-host> ::= "received by" <SP> <host> <SP>

<protocol> ::= "TCP" | "NCP" | "NITS" | "X25" | "INTERNET" |
"ARPANET"

Note: INTERNET = TCP, ARPANET = NCP, and if the <ptcl> is not present INTERNET is assumed.

<daytime> ::= "at" <SP> <date> <SP> <time>

<date> ::= <dd> "-" <mon> "-" <yy>

<time> ::= <hh> ":" <mm> ":" <ss> "-" <zone>

<dd> ::= the one or two decimal integer day of the month in the range 1 to 31.

<mon> ::= "JAN" | "FEB" | "MAR" | "APR" | "MAY" | "JUN" |
"JUL" | "AUG" | "SEP" | "OCT" | "NOV" | "DEC"

<yy> ::= the two decimal integer year of the century in the range 01 to 99.

<hh> ::= the two decimal integer hour of the day in the
range 00 to 24.

<mm> ::= the two decimal integer minute of the hour in the
range 00 to 59.

<ss> ::= the two decimal integer second of the minute in the
range 00 to 59.

<zone> ::= a time zone designator (as in [2]) or "UT" for
Universal Time (the default).

Return Path Example:

Return-Path: <@CHARLIE,@BAKER,JOE@ABLE>

Mail From Example:

Mail-From: ABC received by XYZ at 22-OCT-81 09:23:59-PDT

4.2. SMTP REPLIES

Replies to SMTP commands are devised to ensure the synchronization of requests and actions in the process of mail transfer, and to guarantee that the sender-SMTP always knows the state of the receiver-SMTP. Every command must generate exactly one reply.

The details of the command-reply sequence are made explicit in Section 5.3 on Sequencing and Section 5.4 State Diagrams.

An SMTP reply consists of a three digit number (transmitted as three alphanumeric characters) followed by some text. The number is intended for use by automata to determine what state to enter next; the text is meant for the human user. It is intended that the three digits contain enough encoded information that the sender-SMTP need not examine the text and may either discard it or pass it on to the user, as appropriate. In particular, the text may be receiver-dependent, so there are likely to be varying texts for each reply code. A discussion of the theory of reply codes is given in the Appendix E. Formally, a reply is defined to be the sequence: a three-digit code, <SP>, one line of text, and <CRLF>, or a multiline reply (as defined in Appendix E). Only the EXPN and HELP command are expected to result in multiline replies in normal circumstances, however multiline replies are allowed for any command.

4.2.1. REPLY CODES BY FUNCTION GROUPS

500 Syntax error, command unrecognized
[This may include errors such as command line too long]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented

211 System status, or system help reply
214 Help message
[Information on how to use the receiver or the meaning of a particular non-standard command; this reply is useful only to the human user]

220 <host> Service ready
221 <host> Service closing transmission channel
421 <host> Service not available, closing transmission channel
[This may be a reply to any command if the service knows it must shut down]

250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>
450 Requested mail action not taken: mailbox unavailable
[E.g., mailbox busy]
550 Requested action not taken: mailbox unavailable
[E.g., mailbox not found, no access]
451 Requested action aborted: error in processing
551 User not local; please try <forward-path>
452 Requested action not taken: insufficient system storage
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed
[E.g., mailbox syntax incorrect]
354 Start mail input; end with <CRLF>.<CRLF>
554 Transaction failed

4.2.2. NUMERIC ORDER LIST OF REPLY CODES

211 System status, or system help reply
214 Help message
 [Information on how to use the receiver or the meaning of a
 particular non-standard command; this reply is useful only
 to the human user]
220 <host> Service ready
221 <host> Service closing transmission channel
250 Requested mail action okay, completed
251 User not local; will forward to <forward-path>

354 Start mail input; end with <CRLF>.<CRLF>

421 <host> Service not available, closing transmission channel
 [This may be a reply to any command if the service knows it
 must shut down]
450 Requested mail action not taken: mailbox unavailable
 [E.g., mailbox busy]
451 Requested action aborted: local error in processing
452 Requested action not taken: insufficient system storage

500 Syntax error, command unrecognized
 [This may include errors such as command line too long]
501 Syntax error in parameters or arguments
502 Command not implemented
503 Bad sequence of commands
504 Command parameter not implemented
550 Requested action not taken: mailbox unavailable
 [E.g., mailbox not found, no access]
551 User not local; please try <forward-path>
552 Requested mail action aborted: exceeded storage allocation
553 Requested action not taken: mailbox name not allowed
 [E.g., mailbox syntax incorrect]
554 Transaction failed

4.3. SEQUENCING OF COMMANDS AND REPLIES

The communication between the sender and receiver is intended to be an alternating dialogue, controlled by the sender. As such, the sender issues a command and the receiver responds with a reply. The sender must wait for this response before sending further commands.

One important reply is the connection greeting. Normally, a receiver will send a 220 "Awaiting input" reply when the connection is completed. The sender should wait for this greeting message before sending any commands.

Note: all the greeting type replies have the official name of the server host as the first word following the reply code.

For example,

```
220 <SP> USC-ISIF <SP> Service ready <CRLF>
```

The table below lists alternative success and failure replies for each command. These must be strictly adhered to; a receiver may substitute text in the replies, but the meaning and action implied by the code numbers and by the specific command reply sequence cannot be altered.

COMMAND-REPLY SEQUENCES

Each command is listed with its possible replies. The prefixes used before the possible replies are "P" for preliminary (not used in SMTP), "I" for intermediate, "S" for success, "F" for failure, and "E" for error. The 421 reply (service not available, closing transmission channel) may be given to any command if the SMTP-receiver knows it must shut down. This listing forms the basis for the State Diagrams in Section 4.4.

CONNECTION ESTABLISHMENT

S: 220

F: 421

HELO

S: 250

E: 500, 501, 504, 421

MAIL

S: 250

F: 552, 451, 452

E: 500, 501, 421

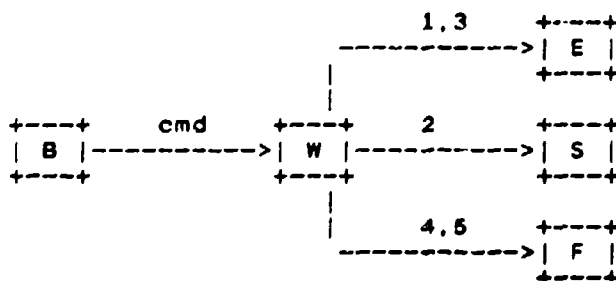
```
RCPT
  S: 250, 251
  F: 550, 551, 552, 553, 450, 451, 452
  E: 500, 501, 421
DATA
  I: 354 -> data -> S: 250
                      F: 552, 554, 451, 452
  F: 451, 554
  E: 500, 501, 421
RSET
  S: 250
  E: 500, 501, 504, 421
SEND
  S: 250
  F: 552, 451, 452
  E: 500, 501, 502, 421
SOML
  S: 250
  F: 552, 451, 452
  E: 500, 501, 502, 421
SAML
  S: 250
  F: 552, 451, 452
  E: 500, 501, 502, 421
VRFY
  S: 250
  F: 550
  E: 500, 501, 502, 504, 421
EXPN
  S: 250
  F: 550
  E: 500, 501, 502, 504, 421
HELP
  S: 211, 214
  E: 500, 501, 502, 504, 421
NOOP
  S: 250
  E: 500, 421
QUIT
  S: 221
  E: 500
```


4.4. STATE DIAGRAMS

Following are state diagrams for a simple-minded SMTP implementation. Only the first digit of the reply codes is used. There is one state diagram for each group of SMTP commands. The command groupings were determined by constructing a model for each command and then collecting together the commands with structurally identical models.

For each command there are three possible outcomes: "success" (S), "failure" (F), and "error" (E). In the state diagrams below we use the symbol B for "begin", and the symbol W for "wait for reply".

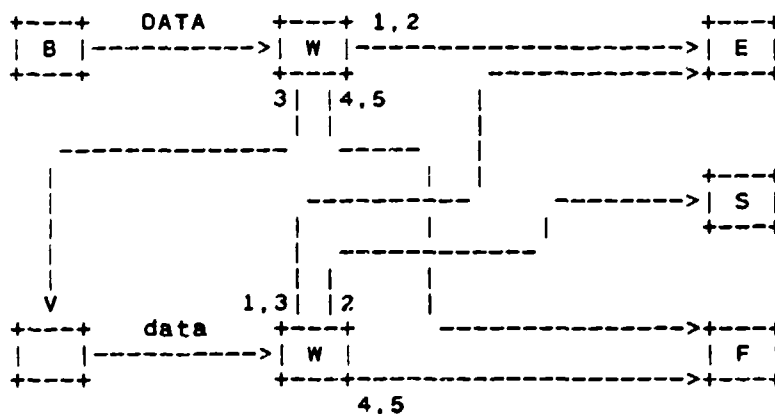
First, the diagram that represents most of the SMTP commands:



This diagram models the commands:

HELO, MAIL, RCPT, RSET, SEND, SOML, SAML, VRFY, EXPN, HELP,
NOOP, QUIT.

A more complex diagram models the DATA command:



Note that the "data" here is a series of lines sent from the sender to the receiver with no response expected until the last line is sent.

4.5. DETAILS

4.5.1. MINIMUM IMPLEMENTATION

In order to make SMTP workable, the following minimum implementation is required for all receivers:

```
COMMANDS -- HELO
              MAIL
              RCPT
              DATA
              RSET
              NOOP
              QUIT
```

4.5.2. TRANSPARENCY

Without some provision for data transparency the character sequence "<CRLF>.<CRLF>" ends the the mail text and cannot be sent by the user. In general, users are not aware of such "forbidden" sequences. To allow all user composed text to be transmitted transparently the following procedures are used.

1. Before sending a line of mail text the sender-SMTP checks the first character of the line. If it is a period, one additional period is inserted at the beginning of the line.
2. When a line of mail text is received by the receiver-SMTP it checks the the line. If the line is composed of a single period it is the end of mail. If the first character is a period and there are other characters on the line, the first character is deleted.

The mail data may contain any of the 128 ASCII characters. All characters are to be delivered to the recipients mailbox including format effectors and other control characters. The 7-bit ASCII codes are transmitted right justified in 8-bit bytes (octets) with the high order bits cleared to zero.

In some systems it may be necessary to transform the data as it is received and stored. This may be necessary for hosts that use a different character set than ASCII as their local character set, or that store data in records rather than strings. If such transforms are necessary, they must be reversible -- especially if such transforms are applied to mail being relayed.

There are several objects that have required minimum maximum sizes. That is every implementation must be able to receive objects of at least these sizes, but must not send objects larger than these sizes.

The maximum total length of a text line including the <CRLF> is 1000 characters (but not counting the leading dot duplicated for transparency).

recipients buffer

The maximum total number of recipients that must be buffered is 100 recipients.

```
.....  
*  
* TO THE MAXIMUM EXTENT POSSIBLE, IMPLEMENTATION *  
* TECHNIQUES WHICH IMPOSE NO LIMITS ON THE LENGTH *  
* OF THESE OBJECTS SHOULD BE USED. *  
*  
.....
```

Errors due to exceeding these limits may be reported by using the reply codes, for example:

500 Line too long.

501 Path too long

552 Too many recipients.

552 Too much mail data.

APPENDIX A

TCP Transport service

The Transmission Control Protocol [3] is used in the ARPA Internet, and in any network following the US DoD standards for internetwork protocols.

Connection Establishment

The SMTP transmission channel is a TCP connection established between the sender process port U and the receiver process port L. This single full duplex connection is used as the transmission channel. This protocol is assigned the service port 25 (31 octal), that is L=25.

Data Transfer

The TCP connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as a 8-bit byte with the high-order bit cleared to zero.

APPENDIX B

NCP Transport service

The ARPANET Host-to-Host Protocol [4] (implemented by the Network Control Program) may be used in the ARPANET.

Connection Establishment

The SMTP transmission channel is established via NCP between the sender process socket U and receiver process socket L. The Initial Connection Protocol [5] is followed resulting in a pair of simplex connections. This pair of connections is used as the transmission channel. This protocol is assigned the contact socket 25 (31 octal), that is L=25.

Data Transfer

The NCP data connections are established in 8-bit byte mode. The SMTP data is 7-bit ASCII characters. Each character is transmitted as a 8-bit byte with the high-order bit cleared to zero.

APPENDIX C

NITS

The Network Independent Transport Service [6] may be used.

Connection Establishment

The SMTP transmission channel is established via NITS between the sender process and receiver process. The sender process executes the CONNECT primitive, and the waiting receiver process executes the ACCEPT primitive.

Data Transfer

The NITS connection supports the transmission of 8-bit bytes. The SMTP data is 7-bit ASCII characters. Each character is transmitted as a 8-bit byte with the high-order bit cleared to zero.

APPENDIX D

X.25 Transport service

It may be possible to use the X.25 service [7] as provided by the Public Data Networks directly, but there are indications that it is too error prone to qualify as a reliable channel. It is suggested that a reliable end-to-end protocol such as TCP be used on top of X.25 connections.

APPENDIX E

Theory of Reply Codes

The three digits of the reply each have a special significance. The first digit denotes whether the response is good, bad or incomplete. An unsophisticated sender-SMTP will be able to determine its next action (proceed as planned, redo, retrench, etc.) by simply examining this first digit. A sender-SMTP that wants to know approximately what kind of error occurred (e.g., mail system error, command syntax error) may examine the second digit, reserving the third digit for the finest gradation of information.

There are five values for the first digit of the reply code:

1yz Positive Preliminary reply

The command has been accepted, but the requested action is being held in abeyance, pending confirmation of the information in this reply. The sender-SMTP should send another command specifying whether to continue or abort the action.

[Note: SMTP does not have any commands that allow this type of reply, and so does not have the continue or abort commands.]

2yz Positive Completion reply

The requested action has been successfully completed. A new request may be initiated.

3yz Positive Intermediate reply

The command has been accepted, but the requested action is being held in abeyance, pending receipt of further information. The sender-SMTP should send another command specifying this information. This reply is used in command sequence groups.

4yz Transient Negative Completion reply

The command was not accepted and the requested action did not occur. However, the error condition is temporary and the action may be requested again. The sender should

return to the beginning of the command sequence (if any). It is difficult to assign a meaning to "transient" when two different sites (receiver- and sender- SMTPs) must agree on the interpretation. Each reply in this category might have a different time value, but the sender-SMTP is encouraged to try again. A rule of thumb to determine if a reply fits into the 4yz or the 5yz category (see below) is that replies are 4yz if they can be repeated without any change in command form or in properties of the sender or receiver. (E.g., the command is repeated identically and the receiver does not put up a new implementation.)

5yz Permanent Negative Completion reply

The command was not accepted and the requested action did not occur. The sender-SMTP is discouraged from repeating the exact request (in the same sequence). Even some "permanent" error conditions can be corrected, so the human user may want to direct the sender-SMTP to reinitiate the command sequence by direct action at some point in the future (e.g., after the spelling has been changed, or the user has altered the account status).

The second digit encodes responses in specific categories:

- x0z Syntax -- These replies refer to syntax errors, syntactically correct commands that don't fit any functional category, and unimplemented or superfluous commands.
- x1z Information -- These are replies to requests for information, such as status or help.
- x2z Connections -- These are replies referring to the transmission channel.
- x3z Unspecified as yet.
- x4z Unspecified as yet.
- x5z Mail system -- These replies indicate the status of the receiver mail system vis-a-vis the requested transfer or other mail system action.

The third digit gives a finer gradation of meaning in each category specified by the second digit. The list of replies

illustrates this. Each reply text is recommended rather than mandatory, and may even change according to the command with which it is associated. On the other hand, the reply codes must strictly follow the specifications in this section. Receiver implementations should not invent new codes for slightly different situations from the ones described here, but rather adapt codes already defined.

For example, a command such as NOOP whose successful execution does not offer the sender-SMTP any new information will return a 250 reply. The response is 502 when the command requests an unimplemented non-site-specific action. A refinement of that is the 504 reply for a command that is implemented, but that requests an unimplemented parameter.

The reply text may be longer than a single line; in these cases the complete text must be marked so the sender-SMTP knows when it can stop reading the reply. This requires a special format to indicate a multiple line reply.

The format for multi-line replies requires that every line, except the last, begin with the reply code, followed immediately by a hyphen, "-" (also known as minus), followed by text. The last line will begin with the reply code, followed immediately by <SP>, optionally some text, and <CRLF>.

For example:

```
123-First line
123-Second line
123-234 text beginning with numbers
123 The last line
```

The sender-SMTP then simply needs to search for the reply code followed by <SP> at the beginning of a line, and ignore all preceding lines.

APPENDIX F

Scenarios

This section presents complete scenarios of several types of SMTP sessions.

A Typical SMTP Transaction Scenario

This SMTP example shows mail sent by Smith at host USC-ISIF, to Jones, Green, and Brown at host BBN-UNIX. Here we assume that host USC-ISIF contacts host BBN-UNIX directly. The mail is accepted for Jones and Brown. Green does not have a mailbox at host BBN-UNIX.

```
R: 220 BBN-UNIX Simple Mail Transfer Service Ready
S: HELO USC-ISIF
R: 250 BBN-UNIX

S: MAIL FROM:<Smith@USC-ISIF>
R: 250 OK

S: RCPT TO:<Jones@BBN-UNIX>
R: 250 OK

S: RCPT TO:<Green@BBN-UNIX>
R: 550 No such user here

S: RCPT TO:<Brown@BBN-UNIX>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 BBN-UNIX Service closing transmission channel
```

Scenario 1

Aborted SMTP Transaction Scenario

R: 220 MIT-Multics Simple Mail Transfer Service Ready
S: HELO ISI-VAXA
R: 250 MIT-Multics

S: MAIL FROM:<Smith@ISI-VAXA>
R: 250 OK

S: RCPT TO:<Jones@MIT-Multics>
R: 250 OK

S: RCPT TO:<Green@MIT-Multics>
R: 550 No such user here

S: RSET
R: 250 OK

S: QUIT
R: 221 MIT-Multics Service closing transmission channel

Scenario 2

Relayed Mail Scenario

Step 1 -- Source Host to Relay Host

R: 220 USC-ISIE Simple Mail Transfer Service Ready
S: HELO MIT-AI
R: 250 USC-ISIE

S: MAIL FROM:<JQP@MIT-AI>
R: 250 OK

S: RCPT TO:<@ISIE,Jones@BBN-VAX>
R: 250 OK

S: DATA
R: 354 Start mail input: end with <CRLF>.<CRLF>
S: Date: 2-Nov-81 22:33:44
S: From: John Q. Public <JQP at MIT-AI>
S: Subject: The Next Meeting of the Board
S: To: Jones at BBN-Vax
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S: John.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE Service closing transmission channel

Step 2 -- Relay Host to Destination Host

```
R: 220 BBN-VAX Simple Mail Transfer Service Ready
S: HELO USC-ISIE
R: 250 BBN-VAX

S: MAIL FROM:<@ISIE.JQP@MIT-AI>
R: 250 OK

S: RCPT TO:<Jones@BBN-VAX>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Mail-From: NCP host MIT-AI received by USC-ISIE at
  2-Nov-81 22:40:10
S: Date: 2-Nov-81 22:33:44
S: From: John Q. Public <JQP at MIT-AI>
S: Subject: The Next Meeting of the Board
S: To: Jones at BBN-Vax
S:
S: Bill:
S: The next meeting of the board of directors will be
S: on Tuesday.
S:
S: John.
S:
R: 250 OK

S: QUIT
R: 221 USC-ISIE Service closing transmission channel
```

Scenario 3

Verifying and Sending Scenario

R: 220 SU-SCORE Simple Mail Transfer Service Ready
S: HELO MIT-MC
R: 250 SU-SCORE

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE>

S: SEND FROM:<EAK@MIT-MC>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE Service closing transmission channel

Scenario 4

Sending and Mailing Scenarios

First the user's name is verified, then an attempt is made to send to the user's terminal. When that fails, the message is mailed to the user's mailbox.

R: 220 SU-SCORE Simple Mail Transfer Service Ready
S: HELO MIT-MC
R: 250 SU-SCORE

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE>

S: SEND FROM:<EAK@MIT-MC>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE>
R: 450 User not active now

S: RSET
R: 250 OK

S: MAIL FROM:<EAK@MIT-MC>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE Service closing transmission channel

Scenario 5

Doing the preceding scenario more efficiently.

```
R: 220 SU-SCORE Simple Mail Transfer Service Ready
S: HELO MIT-MC
R: 250 SU-SCORE

S: VRFY Crispin
R: 250 Mark Crispin <Admin.MRC@SU-SCORE>

S: SOML FROM:<EAK@MIT-MC>
R: 250 OK

S: RCPT TO:<Admin.MRC@SU-SCORE>
R: 250 User not active now, so will do mail.

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 SU-SCORE Service closing transmission channel
```

Scenario 6

Mailing List Scenario

First each of two mailing lists are expanded in separate sessions with different hosts. Then the message is sent to everyone that appeared on either list (but no duplicates) via a relay host.

Step 1 -- Expanding the First List

```
R: 220 MIT-AI Simple Mail Transfer Service Ready
S: HELO SU-SCORE
R: 250 MIT-AI

S: EXPN Example-People
R: 250-<ABC@MIT-MC>
R: 250-Fred Fonebone <Fonebone@ISIQ>
R: 250-Xenon Y. Zither <XYZ@MIT-AI>
R: 250-Quincy Smith <@ISIF,Q-Smith@ISI-VAXA>
R: 250-<joe@foo-unix>
R: 250 <xyz@bar-unix>

S: QUIT
R: 221 MIT-AI Service closing transmission channel
```

Step 2 -- Expanding the Second List

R: 220 MIT-MC Simple Mail Transfer Service Ready
S: HELO SU-SCORE
R: 250 MIT-MC

S: EXPN Interested-Parties
R: 250-Al Calico <ABC@MIT-MC>
R: 250-<XYZ@MIT-AI>
R: 250-Quincy Smith <@ISIF,Q-Smith@ISI-VAXA>
R: 250-<fred@BBN-UNIX>
R: 250 <xyz@bar-unix>

S: QUIT
R: 221 MIT-MC Service closing transmission channel

Step 3 -- Mailing to All via a Relay Host

```
R: 220 USC-ISIE Simple Mail Transfer Service Ready
S: HELO SU-SCORE
R: 250 USC-ISIE

S: MAIL FROM:<Account.Person@SU-SCORE>
R: 250 OK
S: RCPT TO:<@ISIE,ABC@MIT-MC>
R: 250 OK
S: RCPT TO:<@ISIE,Fonebone@ISIQ>
R: 250 OK
S: RCPT TO:<@ISIE,XYZ@MIT-AI>
R: 250 OK
S: RCPT TO:<@ISIE,@ISIF.Q-Smith@ISI-VAXA>
R: 250 OK
S: RCPT TO:<@ISIE,joe@FOO-UNIX>
R: 250 OK
S: RCPT TO:<@ISIE,xyz@BAR-UNIX>
R: 250 OK
S: RCPT TO:<@ISIE,fred@BBN-UNIX>
R: 250 OK

S: DATA
R: 354 Start mail input: end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIE Service closing transmission channel
```

Scenario 7

Forwarding Scenarios

R: 220 USC-ISIF Simple Mail Transfer Service Ready
S: HELO LBL-UNIX
R: 250 USC-ISIF

S: MAIL FROM:<mo@LBL-UNIX>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF>
R: 251 User not local; will forward to <Jones@USC-ISIA>

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIF Service closing transmission channel

Scenario 8

Step 1 -- Trying the Mailbox at the First Host

R: 220 USC-ISIF Simple Mail Transfer Service Ready
S: HELO LBL-UNIX
R: 250 USC-ISIF

S: MAIL FROM:<mo@LBL-UNIX>
R: 250 OK

S: RCPT TO:<fred@USC-ISIF>
R: 251 User not local: will forward to <Jones@USC-ISIA>

S: RSET
R: 250 OK

S: QUIT
R: 221 USC-ISIF Service closing transmission channel

Step 2 -- Delivering the Mail at the Second Host

R: 220 USC-ISIA Simple Mail Transfer Service Ready
S: HELO LBL-UNIX
R: 250 USC-ISIA

S: MAIL FROM:<mo@LBL-UNIX>
R: 250 OK

S: RCPT TO:<Jones@USC-ISIA>
R: OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 USC-ISIA Service closing transmission channel

Scenario 9

Too Many Recipients Scenario

R: 220 BERKELEY Simple Mail Transfer Service Ready
S: HELO USC-ISIF
R: 250 BERKELEY

S: MAIL FROM:<Postel@USC-ISIF>
R: 250 OK

S: RCPT TO:<fabry@BERKELEY>
R: 250 OK

S: RCPT TO:<eric@BERKELEY>
R: 552 Recipient storage full, try again in another transaction

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: MAIL FROM:<Postel@USC-ISIF>
R: 250 OK

S: RCPT TO:<eric@BERKELEY>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CRLF>.<CRLF>
S: Blah blah blah...
S: ...etc. etc. etc.
S: .
R: 250 OK

S: QUIT
R: 221 BERKELEY Service closing transmission channel

Scenario 10

GLOSSARY

ASCII

American Standard Code for Information Interchange [1].

command

A request for a mail service action sent by the sender-SMTP to the receiver-SMTP.

end of mail data indication

A special sequence of characters that indicates the end of the mail data. In particular, the five characters carriage return, line feed, period, carriage return, line feed, in that order.

host

A computer in the internetwork environment on which mailboxes or SMTP processes reside.

line

A line of text ending with a <CRLF>.

mail data

A sequence of ASCII characters of arbitrary length, which conforms to the standard set in the Standard for the Format of ARPA Network Text Messages (RFC 733 [2]).

mailbox

A character string (address) which identifies a user to whom mail is to be sent. Mailbox normally consists of the host and user specifications. The standard mailbox naming convention is defined to be "user@host". Additionally, the "container" in which mail is stored.

receiver-SMTP process

A process which transfers mail in cooperation with a sender-SMTP process. It waits for a connection to be established via the transport service. It receives SMTP commands from the sender-SMTP, sends replies, and performs the specified operations.

reply

A reply is an acknowledgment (positive or negative) sent from receiver to sender via the transmission channel in response to a SMTP command. The general form of a reply is a completion code (including error codes) followed by a text string. The codes are for use by programs and the text is usually intended for human users.

sender-SMTP process

A process which transfers mail in cooperation with a receiver-SMTP process. A local language may be used in the user interface command/reply dialogue. The sender-SMTP initiates the transport service connection. It initiates SMTP commands, receives replies, and governs the transfer of mail.

session

The set of exchanges that occur while the transmission channel is open.

transaction

The set of exchanges required for one message to be transmitted for one or more recipients.

transmission channel

A full-duplex communication path between a sender-SMTP and a receiver-SMTP for the exchange of commands, replies, and mail text.

transport service

Any reliable stream-oriented data communication services. For example, NCP, TCP, NITS.

user

A human being (or a process on behalf of a human being) wishing to obtain mail transfer service. In addition, a recipient of computer mail.

word

A sequence of printing characters.

<CRLF>

The characters carriage return and line feed (in that order).

<SP>

The space character.

REFERENCES

[1] ASCII

ASCII. "USA Code for Information Interchange". United States of America Standards Institute. X3.4. 1968. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[2] RFC 733

Crocker, D., J. Vittal, K. Pogran, and D. Henderson, "Standard for the Format of ARPA Network Text Messages," RFC 733, NIC 41952, November 1977. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[3] TCP

Postel, J., ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, USC/Information Sciences Institute, September 1981.

[4] NCP

McKenzie, A., "Host/Host Protocol for the ARPA Network", NIC 8246, January 1972. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[5] Initial Connection Protocol

Postel, J., "Official Initial Connection Protocol", NIC 7101, 11 June 1971. Also in: Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.

[6] NITS

PSS/SQ3, "A Network Independent Transport Service", Study Group 3, The Post Office PSS Users Group, February 1980. Available from the DCPU, National Physical Laboratory, Teddington, UK.

[7] X.25

CCITT, "Recommendation X.25 - Interface Between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for Terminals Operating in the Packet Mode on Public Data Networks," CCITT Orange Book, Vol. VIII.2, International Telephone and Telegraph Consultative Committee, Geneva, 1976.

RFC-783

TRIVIAL FILE TRANSFER PROTOCOL

June 1981

(379)

Network Working Group
Request for Comments: 783

Updates: IEN 133

K. R. Sollins
MIT
June, 1981

THE TFTP PROTOCOL

(REVISION 2)

SUMMARY

TFTP is a very simple protocol used to transfer files. It is from this that its name comes, Trivial File Transfer Protocol or TFTP. Each nonterminal packet is acknowledged separately. This document describes the protocol and its types of packets. The document also explains the reasons behind some of the design decisions.

ACKNOWLEDGEMENTS

The protocol was originally designed by Noel Chiappa, and was redesigned by him, Bob Baldwin and Dave Clark, with comments from Steve Szymanski. The current revision of the document includes modifications stemming from discussions with and suggestions from Larry Allen, Noel Chiappa, Dave Clark, Geoff Cooper, Mike Greenwald, Liza Martin, David Reed, Craig Milo Rogers (of UCS-ISI), Kathy Yellick, and the author. The acknowledgement and retransmission scheme was inspired by TCP, and the error mechanism was suggested by PARC's EFTP abort message.

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract number N00014-75-C-0661.

1. Purpose

TFTP is a simple protocol to transfer files, and therefore was named the Trivial File Transfer Protocol or TFTP. It has been implemented on top of the Internet User Datagram protocol (UDP or Datagram) [2] so it may be used to move files between machines on different networks implementing UDP. (This should not exclude the possibility of implementing TFTP on top of other datagram protocols.) It is designed to be small and easy to implement. Therefore, it lacks most of the features of a regular FTP. The only thing it can do is read and write files (or mail) from/to a remote server. It cannot list directories, and currently has no provisions for user authentication. In common with other Internet protocols, it passes 8 bit bytes of data.

Three modes of transfer are currently supported: netascii¹; octet², raw 8 bit bytes; mail, netascii characters sent to a user rather than a file. Additional modes can be defined by pairs of cooperating hosts.

¹ This is ascii as defined in "USA Standard Code for Information Interchange" [1] with the modifications specified in "Telnet Protocol Specification" [3]. Note that it is 8 bit ascii. The term "netascii" will be used throughout this document to mean this particular version of ascii.

² This replaces the "binary" mode of previous versions of this document.

2. Overview of the Protocol

Any transfer begins with a request to read or write a file, which also serves to request a connection. If the server grants the request, the connection is opened and the file is sent in fixed length blocks of 512 bytes. Each data packet contains one block of data, and must be acknowledged by an acknowledgment packet before the next packet can be sent. A data packet of less than 512 bytes signals termination of a transfer. If a packet gets lost in the network, the intended recipient will timeout and may retransmit his last packet (which may be data or an acknowledgment), thus causing the sender of the lost packet to retransmit that lost packet. The sender has to keep just one packet on hand for retransmission, since the lock step acknowledgment guarantees that all older packets have been received. Notice that both machines involved in a transfer are considered senders and receivers. One sends data and receives acknowledgments, the other sends acknowledgments and receives data.

Most errors cause termination of the connection. An error is signalled by sending an error packet. This packet is not acknowledged, and not retransmitted (i.e., a TFTP server or user may terminate after sending an error message), so the other end of the connection may not get it. Therefore timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events: not being able to satisfy the request (e.g., file not found, access violation, or no such user), receiving a packet which cannot be explained by a delay or duplication in the network (e.g. an incorrectly

formed packet). and losing access to a necessary resource (e.g., disk full or access denied during a transfer).

TFTP recognizes only one error condition that does not cause termination, the source port of a received packet being incorrect. In this case, an error packet is sent to the originating host.

This protocol is very restrictive, in order to simplify implementation. For example, the fixed length blocks make allocation straight forward, and the lock step acknowledgement provides flow control and eliminates the need to reorder incoming data packets.

3. Relation to other Protocols

As mentioned TFTP is designed to be implemented on top of the Datagram protocol. Since Datagram is implemented on the Internet protocol, packets will have an Internet header, a Datagram header, and a TFTP header. Additionally, the packets may have a header (LNI, ARPA header, etc.) to allow them through the local transport medium. As shown in Figure 3-1, the order of the contents of a packet will be: local medium header, if used, Internet header, Datagram header, TFTP header, followed by the remainder of the TFTP packet. (This may or may not be data depending on the type of packet as specified in the TFTP header.) TFTP does not specify any of the values in the Internet header. On the other hand, the source and destination port fields of the Datagram header (its format is given in the appendix) are used by TFTP and the length field reflects the size of the TFTP packet. The transfer identifiers (TID's)

used by TFTP are passed to the Datagram layer to be used as ports; therefore they must be between 0 and 65,535. The initialization of TID's is discussed in the section on initial connection protocol.

The TFTP header consists of a 2 byte opcode field which indicates the packet's type (e.g., DATA, ERROR, etc.) These opcodes and the formats of the various types of packets are discussed further in the section on TFTP packets.

Figure 3-1: Order of Headers



4. Initial Connection Protocol

A transfer is established by sending a request (WRQ to write onto a foreign file system, or RRQ to read from it), and receiving a positive reply, an acknowledgment packet for write, or the first data packet for read. In general an acknowledgment packet will contain the block number of the data packet being acknowledged. Each data packet has associated with it a block number; block numbers are consecutive and begin with one. Since the positive response to a write request is an acknowledgment packet, in this special case the block number will be zero. (Normally, since an acknowledgment packet is acknowledging a data packet, the acknowledgment packet will contain the block number of the data packet being acknowledged.) If the reply is an error packet, then the request has been denied.

In order to create a connection, each end of the connection chooses a TID for itself, to be used for the duration of that connection. The TID's chosen for a connection should be randomly chosen, so that the probability that the same number is chosen twice in immediate succession is very low. Every packet has associated with it the two TID's of the ends of the connection, the source TID and the destination TID. These TID's are handed to the supporting UDP (or other datagram protocol) as the source and destination ports. A requesting host chooses its source TID as described above, and sends its initial request to the known TID 69 decimal (105 octal) on the serving host. The response to the request, under normal operation, uses a TID chosen by the server as its source TID and the TID chosen for the previous message by the requestor as its destination TID. The two chosen TID's are then used for the remainder of the transfer.

As an example, the following shows the steps used to establish a connection to write a file. Note that WRQ, ACK, and DATA are the names of the write request, acknowledgment, and data types of packets respectively. The appendix contains a similar example for reading a file.

1. Host A sends a "WRQ" to host B with source= A's TID, destination= 69.
2. Host B sends a "ACK" (with block number= 0) to host A with source= B's TID, destination= A's TID.

At this point the connection has been established and the first data packet can be sent by Host A with a sequence number of 1. In the next step, and in all succeeding steps, the hosts should make sure that the source TID matches the value that was agreed on in steps 1 and 2. If a source TID does not match, the packet should be discarded as erroneously sent from somewhere else. An error packet should be sent to the source of the incorrect packet, while not disturbing the transfer. This can be done only if the TFTP in fact receives a packet with an incorrect TID. If the supporting protocols do not allow it, this particular error condition will not arise.

The following example demonstrates a correct operation of the protocol in which the above situation can occur. Host A sends a request to host B. Somewhere in the network, the request packet is duplicated, and as a result two acknowledgments are returned to host A, with different TID's chosen on host B in response to the two requests. When the first response arrives, host A continues the connection. When the second response to the request arrives, it should be rejected, but there is no reason to terminate the first connection. Therefore, if different TID's are chosen for the two connections on host B and host A checks the source TID's of the messages it receives, the first connection can be maintained while the second is rejected by returning an error packet.

5. TFTP Packets

TFTP supports five types of packets, all of which have been mentioned above:

opcode	operation
1	Read request (RRQ)
2	Write request (WRQ)
3	Data (DATA)
4	Acknowledgment (ACK)
5	Error (ERROR)

The TFTP header of a packet contains the opcode associated with that packet.

Figure 5-1: RRQ/WRQ packet

2 bytes	string	1 byte	string	1 byte
Opcode	Filename	0		0

RRQ and WRQ packets (opcodes 1 and 2 respectively) have the format shown in Figure 5-1. The filename is a sequence of bytes in netascii terminated by a zero byte. The mode field contains the string "netascii", "octet", or "mail" (or any combination of upper and lower case, such as "NETASCII", "NetAscii", etc.) in netascii indicating the three modes defined in the protocol. A host which receives netascii mode data must translate the data to the local format. Octet mode is used to transfer a file that is in the 8-bit format of the machine from which the file is being transferred. It is assumed that each type of machine has a single 8-bit format that is more common, and that that format is

chosen. For example, on a DEC-20, a 36 bit machine, this is four 8-bit bytes to a word with four bits of breakage. If a host receives a octet file and then returns it, the returned file must be identical to the original. Mail mode uses the name of a mail recipient in place of a file and must begin with a WRQ. Otherwise it is identical to netascii mode. The mail recipient string should be of the form "username" or "username@hostname". If the second form is used, it allows the option of mail forwarding by a relay computer.

The discussion above assumes that both the sender and recipient are operating in the same mode, but there is no reason that this has to be the case. For example, one might build a storage server. There is no reason that such a machine needs to translate netascii into its own form of text. Rather, the sender might send files in netascii, but the storage server might simply store them without translation in 8-bit format. Another such situation is a problem that currently exists on DEC-20 systems. Neither netascii nor octet accesses all the bits in a word. One might create a special mode for such a machine which read all the bits in a word, but in which the receiver stored the information in 8-bit format. When such a file is retrieved from the storage site, it must be restored to its original form to be useful, so the reverse mode must also be implemented. The user site will have to remember some information to achieve this. In both of these examples, the request packets would specify octet mode to the foreign host, but the local host would be in some other mode. No such machine or application specific modes have been specified in TFTP, but one would be compatible with this specification.

It is also possible to define other modes for cooperating pairs of hosts, although this must be done with care. There is no requirement that any other hosts implement these. There is no central authority that will define these modes or assign them names.

Figure 5-2: DATA packet

2 bytes	2 bytes	n bytes
Opcode	Block #	Data

Data is actually transferred in DATA packets depicted in Figure 5-2. DATA packets (opcode = 3) have a block number and data field. The block numbers on data packets begin with one and increase by one for each new block of data. This restriction allows the program to use a single number to discriminate between new packets and duplicates. The data field is from zero to 512 bytes long. If it is 512 bytes long, the block is not the last block of data; if it is from zero to 511 bytes long, it signals the end of the transfer. (See the section on Normal Termination for details.)

All packets other than those used for termination are acknowledged individually unless a timeout occurs. Sending a DATA packet is an acknowledgment for the ACK packet of the previous DATA packet. The WRQ and DATA packets are acknowledged by ACK or ERROR packets, while RRQ and

Figure 5-3: ACK packet

2 bytes	2 bytes
Opcode	Block #

ACK packets are acknowledged by DATA or ERROR packets. Figure 5-3 depicts an ACK packet; the opcode is 4. The block number in an ACK echoes the block number of the DATA packet being acknowledged. A WRQ is acknowledged with an ACK packet having a block number of zero.

Figure 5-4: ERROR packet

2 bytes	2 bytes	string	1 byte
Opcode	ErrorCode	ErrMsg	0

An ERROR packet (opcode 5) takes the form depicted in Figure 5-4. An ERROR packet can be the acknowledgment of any other type of packet. The error code is an integer indicating the nature of the error. A table of values and meanings is given in the appendix. (Note that several error codes have been added to this version of this document.) The error message is intended for human consumption, and should be in netascii. Like all other strings, it is terminated with a zero byte.

6. Normal Termination

The end of a transfer is marked by a DATA packet that contains between 0 and 511 bytes of data (i.e. Datagram length < 516). This packet is acknowledged by an ACK packet like all other DATA packets. The host acknowledging the final DATA packet may terminate its side of the connection on sending the final ACK. On the other hand, dallying is encouraged. This means that the host sending the final ACK will wait for a while before terminating in order to retransmit the final ACK if it has been lost. The acknowledger will know that the ACK has been lost if it receives the final DATA packet again. The host sending the last DATA must retransmit it until the packet is acknowledged or the sending host times out. If the response is an ACK, the transmission was completed successfully. If the sender of the data times out and is not prepared to retransmit any more, the transfer may still have been completed successfully, after which the acknowledger or network may have experienced a problem. It is also possible in this case that the transfer was unsuccessful. In any case, the connection has been closed.

7. Premature Termination

If a request can not be granted, or some error occurs during the transfer, then an ERROR packet (opcode 5) is sent. This is only a courtesy since it will not be retransmitted or acknowledged, so it may never be received. Timeouts must also be used to detect errors.

I. Appendix

Order of Headers

				2 bytes			
	Local Medium		Internet		Datagram		TFTP Opcode

TFTP Formats

Type	Op #	Format without header			
	2 bytes	string	1 byte	string	1 byte
RRQ/ WRQ	01/02	Filename	0	Mode	0
	2 bytes	2 bytes	n bytes		
DATA	03	Block #	Data		
	2 bytes	2 bytes			
ACK	04	Block #			
	2 bytes	2 bytes	string	1 byte	
ERROR	05	ErrorCode	ErrMag	0	

Initial Connection Protocol for reading a file

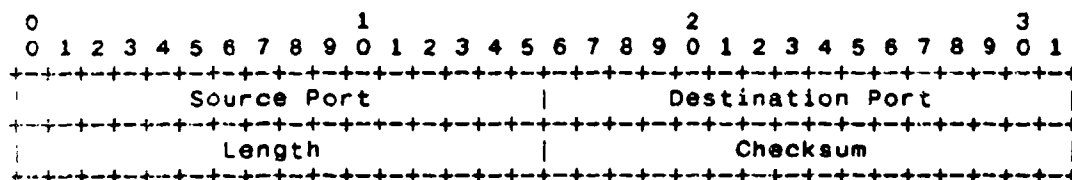
1. Host A sends a "RRQ" to host B with source= A's TID, destination= 69.
2. Host B sends a "DATA" (with block number= 1) to host A with source= B's TID, destination= A's TID.

Error Codes

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

Internet User Datagram Header [2]³

Format



Values of Fields

- Source Port Picked by originator of packet.
- Dest. Port Picked by destination machine (69 for RRQ or WRQ).
- Length Number of bytes in packet after Datagram header.
- Checksum Reference 2 describes rules for computing checksum.⁴
Field contains zero if unused.

Note: TFTP passes transfer identifiers (TID's) to the Internet User Datagram protocol to be used as the source and destination ports.

³ This has been included only for convenience. TFTP need not be implemented on top of the Internet User Datagram Protocol.

⁴ The implementor of this should be sure that the correct algorithm is used here.

References

- [1] USA Standard Code for Information Interchange, USASI X3.4-1968.
- [2] Postel, Jon., "User Datagram Protocol," RFC768, August 28, 1980.
- [3] "Telnet Protocol Specification," RFC764, June, 1980.

IEN-116

NAME SERVER PROTOCOL

August 1979

(397)

IEN 116

J. Postel
ISI
August 1979

Obsoletes: 89, 61

INTERNET NAME SERVER

INTRODUCTION

This memo defines the procedure to access an Internet Name Server. Such a server provides the actual addresses of hosts in the internet when supplied with a host name. An Internet Name Server is a dynamic name-to-number translation service.

This server utilizes the User Datagram Protocol (UDP) [2], which in turn calls on the Internet Protocol (IP) [3].

NAME SYNTAX

It is strongly recommended that the use of host names in programs be consistent for both input and output across all hosts. To promote such consistency of the internet level, the following syntax is specified:

The SYNTAX of names as presented to the user and as entered by the user is:

! NET ! REST

where:

NET is a network name or number as defined in "Assigned Numbers" [1]

and

REST is a host name within that network expressed as a character string or as a number. When a number is used, it is expressed in decimal and is prefixed with a sharp sign (e.g., #1234).

Note that this syntax has minimal impact on the allowable character strings for host names within a network. The only restriction is that a REST string cannot begin with an exclamation point (!).

The !NET! may be omitted when specifying a host in the local network. That is "!" indicates the network portion of a name string.

BASIC NAME SERVER

To aid in the translation of names to internet addresses, several name server processes will be provided. The name server process will accept a name in the above form and will return a name, address pair.

The name server processes will have well-known addresses; addresses that are constant over long periods of time and published in documents such as "Assigned Numbers" [1].

A request sent to a name server is sent as a user datagram [2] with the following content:



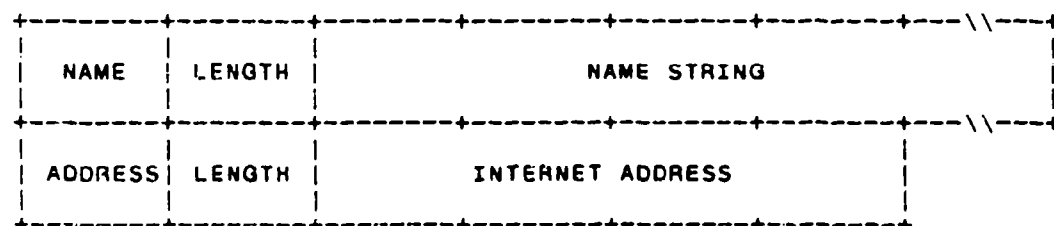
where:

NAME is a one octet code indicating that the following is a name.

LENGTH is a one octet count of the number of octets in the name string, and

NAME STRING is an ASCII character string of the form ! NET ! REST.

A reply to a successful translation is sent as a user datagram with the following content:



where:

ADDRESS is a one octet code indicating that the following is an internet address,

LENGTH is a one octet count (=4) of the length of the internet address, and

INTERNET ADDRESS is the internet address.

Actually a particular name might map to several internet addresses. in this case the response would include a list of internet addresses.

When a name is not found, an error is reported via a user datagram as follows:

NAME	LENGTH	NAME STRING	
ERROR	LENGTH	ERROR CODE	ERROR STRING

where:

ERROR CODE specifies the error.

ERROR STRING explains the error.

Error Codes

The following error codes are defined:

CODE	MEANING
0	Undetermined or undefined error
1	Name not found
2	Improper name syntax

Communication with a Name Server Process

Communication with a name server process is via user datagrams. User datagrams do not guarantee reliable communication. Thus, some requests or replies may be lost.

The name server process is a transaction oriented process; furthermore, the nature of the transactions allows them to be processed in any order and even to be duplicated. This allows the use of a very simple communication protocol.

If a request is made to the name server process and no response is received within a reasonable time, then the requester should make the request again. This recovers from communication errors which cause the loss of either the request or the reply.

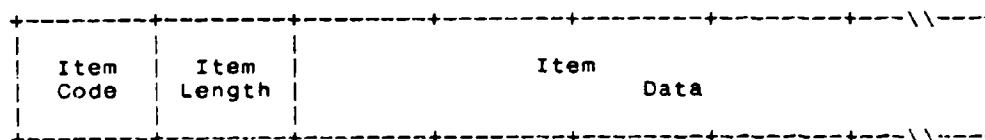
In order to use this simple strategy, care must be taken to allow replies to be properly matched with requests. The name server process does this by including in each reply a copy of the entire request.

The user datagram protocol does provide a checksum for the detection of errors.

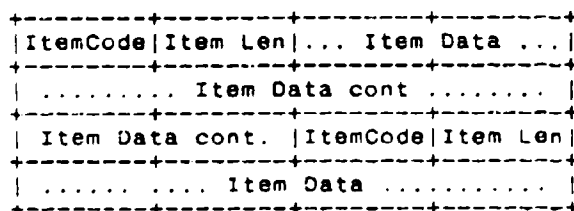
Format

The requests and replies to and from a name server process are encoded as "items". An item consists of an item-code an item-length and the item-data. The item-length includes in its count the item-count and the item-length octets.

Item := Item-Code Item-Length Item-Data



A request is typically one item, and a reply is typically two items.



August 1979
IEN 116

Internet Name Server

Item Code Value Assignments:

NAME = 1

ADDRESS = 2

ERROR = 3

Example

a typical request:

1	12	!	A
R	P	A	!
I	S	I	B

and the reply:

1	12	!	A
R	P	A	!
I	S	I	B
2	6	10	3
0	52		

EXTENDED NAME SERVER

Several extensions have been proposed [4], the following two are adopted: partially specified names, and a service field.

In the first extension partially specified names are allowed and are indicated by the use of "wild card" fields or characters.

Wild Card Field	Meaning
•	All
~	Local (Same as that of the requestor)

Wild Card Character	Meaning
•	Any substring

Examples:

!~!* all hosts on the net of the requestor.

!*!SRI* all hosts with names whose first three characters
 are SRI on all nets

In general, there are three cases for each of the net and host fields. Using the symbols N for named network and H for named host the 9 cases are:

!~!~ local net, local host

!~!* local net, all hosts

!~!H local net, named host

!*!~ all nets, local host

!*!* all nets, all hosts

!*!H all nets, named host

!N!~ named net, local host

!N!* named net, all hosts

!N!H named net, named host

August 1979
IEN 116

Internet Name Server

When such a request is processed and the result is more than one name/address pair, the response is all the pairs.

Examples:

1)

request:

!ARPA!ISI•

response:

!ARPA!ISIA 10 1 0 22

!ARPA!ISIB 10 3 0 52

!ARPA!ISIC 10 2 0 22

!ARPA!ISID 10 3 0 22

!ARPA!ISIE 10 1 0 52

2)

request:

!~!SRI-R2D2

response:

!ARPA!SRI-R2D2 10 3 0 51

!SF-PR-1!SRI-R2D2 2 0 0 11

3)

request:

!•!ISIA

response:

!ARPA!ISIA 10 1 0 22

The second extension is that a third field may be appended to the name. This is the SERVICE field.

! NET ! HOST ! SERVICE

To reply to a request of this form the name server must provide the internet address (net and host), the protocol number, and the port number.

NAME	LENGTH	NAME STRING
ADDRESS	LENGTH	INTERNET ADDRESS
PROTOCOL	PORT	

Examples:

1)

request:

! ARPA ! ISIA ! TELNET

response:

! ARPA ! ISIA ! TELNET 10 1 0 22 6 0 23

2)

request:

! ARPA ! * ! NAME-SERVER

response:

! ARPA ! SRI-KL ! NAME-SERVER 10 1 0 2 17 42

August 1979
IEN 116
References

Internet Name Server

References

- [1] J. Postel. "Assigned Numbers," IEN 117, USC/Information Sciences Institute, August 1979.
- [2] J. Postel. "User Datagram Protocol," IEN 88, USC/Information Sciences Institute, May 1979.
- [3] J. Postel. "Internet Protocol," IEN 111, USC/Information Sciences Institute, August 1979.
- [4] J. Pickens, E. Feinler, and J. Mathis. "The NIC Name Server -- A Datagram Based Information Utility." Proceedings of the Fourth Berkeley Conference on Distributed Data Management and Computer Networks, pp. 275-283, August 1979.

Acknowledgments

John Pickens contributed the ideas for the Extended Name Server.

APPENDICES

ASSIGNED NUMBERS

PRE-EMPTION

SERVICE MAPPINGS

ADDRESS MAPPINGS

**MAIL HEADER
FORMAT STANDARDS**

C/30 INFORMATION

HOST TABLE SPECIFICATION

HOSTNAMES SERVER

NICNAME/WHOIS

RFC-790

ASSIGNED NUMBERS

September 1981

(409)

Network Working Group
Request for Comments: 790

J. Postel
ISI
September 1981

Obsoletes RFCs: 776, 770, 762, 758,
755, 750, 739, 604, 503, 433, 349
Obsoletes IENS: 127, 117, 93

ASSIGNED NUMBERS.

This Network Working Group Request for Comments documents the currently assigned values from several series of numbers used in network protocol implementations. This RFC will be updated periodically, and in any case current information can be obtained from Jon Postel. The assignment of numbers is also handled by Jon. If you are developing a protocol or application that will require the use of a link, socket, port, protocol, or network number please contact Jon to receive a number assignment.

Jon Postel
USC - Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90291

phone: (213) 822-1511

ARPANET mail: POSTEL@ISIF

Most of the protocols mentioned here are documented in the RFC series of notes. The more prominent and more generally used are documented in the Protocol Handbook [17] prepared by the Network Information Center (NIC). Some of the items listed are undocumented. In all cases the name and mailbox of the responsible individual is indicated. In the lists that follow, a bracketed entry, e.g., [17,iii], at the right hand margin of the page indicates a reference for the listed protocol, where the number cites the document and the "iii" cites the person.

Network Numbers

Network Numbers

Network Numbers



Network Numbers

Network Numbers



Network Numbers

Network Numbers



Network Numbers

Network Numbers

Network Numbers

Network Numbers

numbers. The class a networks will have nnn.nnn.nnn.nnn, the class b networks will have nnn.nnn.nnn.nnn, and the class c networks will have nnn.nnn.nnn.nnn, where nnn represents part or all of a network number and rrr represents part or all of a local address or rest field.

Assigned Network Numbers

Class A Networks

Internet Address	Name	Network	References
000.nnn.nnn.nnn		Reserved	[JBP]
001.nnn.nnn.nnn	BBN-PR	BBN Packet Radio Network	[DCA2]
002.nnn.nnn.nnn	SF-PR-1	SF Packet Radio Network (1)	[JEM]
003.nnn.nnn.nnn	BBN-RCC	BBN RCC Network	[SGC]
004.nnn.nnn.nnn	SATNET	Atlantic Satellite Network	[DM11]
005.nnn.nnn.nnn	SILL-PR	Ft. Sill Packet Radio Network	[JEM]
006.nnn.nnn.nnn	SF-PR-2	SF Packet Radio Network (2)	[JEM]
007.nnn.nnn.nnn	CHAOS	MIT CHAOS Network	[MOON]
008.nnn.nnn.nnn	CLARKNET	SATNET subnet for Clarksburg	[DM11]
009.nnn.nnn.nnn	BRAGG-PR	Ft. Bragg Packet Radio Net	[JEM]
010.nnn.nnn.nnn	ARPANET	ARPANET	[17.1,VGC]
011.nnn.nnn.nnn	UCLNET	University College London	[PK]
012.nnn.nnn.nnn	CYCLADES	CYCLADES	[VGC]
013.nnn.nnn.nnn		Unassigned	[JBP]
014.nnn.nnn.nnn	TELENET	TELENET	[VGC]
015.nnn.nnn.nnn	EPSS	British Post Office EPSS	[PK]
016.nnn.nnn.nnn	DATAPAC	DATAPAC	[VGC]
017.nnn.nnn.nnn	TRANSPAC	TRANSPAC	[VGC]
018.nnn.nnn.nnn	LCSNET	MIT LCS Network	[43.10,DDC2]
019.nnn.nnn.nnn	TYMNET	TYMNET	[VGC]
020.nnn.nnn.nnn	DC-PR	D.C. Packet Radio Network	[VGC]
021.nnn.nnn.nnn	EDN	DCEC EDN	[EC5]
022.nnn.nnn.nnn	DIALNET	DIALNET	[26.16,MRC]
023.nnn.nnn.nnn	MITRE	MITRE Cablenet	[44,APS]
024.nnn.nnn.nnn	BBN-LOCAL	BBN Local Network	[SGC]
025.nnn.nnn.nnn	RSRE-PPSN	RSRE / PPSN	[BD2]
026.nnn.nnn.nnn	AUTODIN-II	AUTODIN II	[EC5]
027.nnn.nnn.nnn	NOSC-LCCN	NOSC / LCCN	[KTP]
028.nnn.nnn.nnn	WIDEBAND	Wide Band Satellite Network	[CJW2]
029.nnn.nnn.nnn	DCN-COMSAT	COMSAT Dist. Comp. Network	[DLM1]
030.nnn.nnn.nnn	DCN-UCL	UCL Dist. Comp. Network	[PK]
031.nnn.nnn.nnn	BBN-SAT-TEST	BBN SATNET Test Network	[DM11]
032.nnn.nnn.nnn	UCL-CR1	UCL Cambridge Ring 1	[PK]
033.nnn.nnn.nnn	UCL-CR2	UCL Cambridge Ring 2	[PK]
034.nnn.nnn.nnn	MATNET	Mobile Access Terminal Net	[DM11]
035.nnn.nnn.nnn	NULL	UCL/RSRE Null Network	[BD2]

Network Numbers

036.rrr.rrr.rrr	SU-NET	Stanford University Ethernet	[MRC]
037.rrr.rrr.rrr	DECNET	Digital Equipment Network	[DRL]
038.rrr.rrr.rrr	DECNET-TEST	Test Digital Equipment Net	[DRL]
039.rrr.rrr.rrr	SRINET	SRI Local Network	[GEOF]
040.rrr.rrr.rrr	CISLNET	CISL Multics Network	[CH2]
041.rrr.rrr.rrr	BBN-LN-TEST	BBN Local Network Testbed	[KTP]
042.rrr.rrr.rrr	SINET	LLL-SI-NET	[EAK]
043.rrr.rrr.rrr	INTELPOST	COMSAT INTELPOST	[DLM1]
044.rrr.rrr.rrr	AMPRNET	Amature Radio Experiment Net	[HM]
044.rrr.rrr.rrr-126.rrr.rrr.rrr		Unassigned	[JBP]
127.rrr.rrr.rrr		Reserved	[JBP]

Class B Networks

Internet Address	Name	Network	References
128.000.rrr.rrr		Reserved	[JBP]
128.001.rrr.rrr-128.254.rrr.rrr		Unassigned	[JBP]
191.255.rrr.rrr		Reserved	[JBP]

Class C Networks

Internet Address	Name	Network	References
192.000.001.rrr		Reserved	[JBP]
192.000.001.rrr-223.255.254.rrr		Unassigned	[JBP]
223.255.255.rrr		Reserved	[JBP]

Other Reserved Internet Addresses

Internet Address	Name	Network	References
224.000.000.000-255.255.255.255		Reserved	[JBP]

Internet Version Numbers

ASSIGNED INTERNET VERSION NUMBERS

In the Internet Protocol (IP) [33] there is a field to identify the version of the internetwork general protocol. This field is 4 bits in size.

Assigned Internet Version Numbers

Decimal	Octal	Version	References
0	0	Reserved	[JBP]
1-3	1-3	Unassigned	[JBP]
4	4	Internet Protocol	[33, JBP]
5	5	ST Datagram Mode	[20, JWF]
6-14	6-16	Unassigned	[JBP]
15	17	Reserved	[JBP]

Internet Protocol Numbers

ASSIGNED INTERNET PROTOCOL NUMBERS

In the Internet Protocol (IP) [33] there is a field, called Protocol, to identify the the next level protocol. This is an 8 bit field.

Assigned Internet Protocol Numbers

Decimal	Octal	Protocol Numbers	References
0	0	Reserved	[JBP]
1	1	ICMP	[53, JBP]
2	2	Unassigned	[JBP]
3	3	Gateway-to-Gateway	[48, 49, VMS]
4	4	CMCC Gateway Monitoring Message	[18, 19, DFP]
5	5	ST	[20, JWF]
6	6	TCP	[34, JBP]
7	7	UCL	[PK]
8	10	Unassigned	[JBP]
9	11	Secure	[VGC]
10	12	BBN RCC Monitoring	[VMS]
11	13	NVP	[12, DC]
12	14	PUP	[4, EAT3]
13	15	Pluribus	[RDB2]
14	16	Telenet	[ROB2]
15	17	XNET	[25, JFH2]
16	20	Chaos	[MOON]
17	21	User Datagram	[42, JBP]
18	22	Multiplexing	[13, JBP]
19	23	DCN	[DLM1]
20	24	TAC Monitoring	[55, RH6]
21-62	25-76	Unassigned	[JBP]
63	77	any local network	[JBP]
64	100	SATNET and Backroom EXPAK	[DM11]
65	101	MIT Subnet Support	[NC3]
66-68	102-104	Unassigned	[JBP]
69	105	SATNET Monitoring	[DM11]
70	106	Unassigned	[JBP]
71	107	Internet Packet Core Utility	[DM11]
72-75	110-113	Unassigned	[JBP]
76	114	Backroom SATNET Monitoring	[DM11]
77	115	Unassigned	[JBP]
78	116	WIDEBAND Monitoring	[DM11]
79	117	WIDEBAND EXPAK	[DM11]
80-254	120-376	Unassigned	[JBP]
255	377	Reserved	[JBP]

Port or Socket Numbers

ASSIGNED PORT or SOCKET NUMBERS

Ports are used in the TCP [34] and sockets are used in the AHHP [28,17] to name the ends of logical connections which carry long term conversations. For the purpose of providing services to unknown callers a service contact socket is defined. This list specifies the port or socket used by the server process as its contact socket. In the AHHP an Initial Connection Procedure ICP [39,17] is used between the user process and the server process to make the initial contact and establish the long term connections leaving the contact socket free to handle other callers. In the TCP no ICP is necessary since a port may engage in many simultaneous connections.

To the extent possible these same port assignments are used with UDP [42].

The assigned ports/sockets use a small part of the possible port/socket numbers. The assigned ports/sockets have all except the low order eight bits cleared to zero. The low order eight bits are specified here.

Socket Assignments:

General Assignments:

Decimal	Octal	Description
0-63	0-77	Network Wide Standard Function
64-131	100-203	Hosts Specific Functions
132-223	204-337	Reserved for Future Use
224-255	340-377	Any Experimental Function

Port or Socket Numbers

Specific Assignments:

Network Standard Functions

Decimal	Octal	Description	References
1	1	Old Telnet	[40, JBP]
3	3	Old File Transfer	[27, 11, 24, JBP]
5	5	Remote Job Entry	[6, 17, JBP]
7	7	Echo	[35, JBP]
9	11	Discard	[32, JBP]
11	13	Who is on or SYSTAT	[JBP]
13	15	Date and Time	[JBP]
15	17	Who is up or NETSTAT	[JBP]
17	21	Short Text Message	[JBP]
19	23	Character generator or TTYTST	[31, JBP]
21	25	New File Transfer	[36, JBP]
23	27	New Telnet	[41, JBP]
25	31	SMTP	[54, JBP]
27	33	NSW User System w/COMPASS FE	[14, RHT]
29	35	MSG-3 ICP	[29, RHT]
31	37	MSG-3 Authentication	[29, RHT]
33	41	Unassigned	[JBP]
35	43	IO Station Spooler	[JBP]
37	45	Time Server	[22, JBP]
39	47	Unassigned	[JBP]
41	51	Graphics	[46, 17, JBP]
42	52	Name Server	[38, JBP]
43	53	WhoIs	[JAKE]
45	55	Message Processing Module	[37, JBP]
47	57	NI FTP	[50, CJB]
49	61	RAND Network Graphics Conference	[30, MO2]
51	63	Message Generator Control	[52, DFP]
53	65	AUTODIN II FTP	[21, EC5]
55	67	ISI Graphics Language	[3, RB6]
57	71	MTP	[45, JBP]
59	73	New MIT Host Status	[SWG]
61-63	75-77	Unassigned	[JBP]

Port or Socket Numbers

Host Specific Functions

Decimal	Octal	Description	References
65	101	Unassigned	[JBP]
67	103	Datacomputer at CCA	[8,JZS]
69	105	Unassigned	[JBP]
69	105	Trivial File Transfer	[47,KRS]
71	107	NETRJS (EBCDIC) at UCLA-CCN	[5,17,RTB]
73	111	NETRJS (ASCII-68) at UCLA-CCN	[5,17,RTB]
75	113	NETRJS (ASCII-63) at UCLA-CCN	[5,17,RTB]
77	115	any private RJE server	[JBP]
79	117	Name or Finger	[23,17,KLH]
81	121	Unassigned	[JBP]
83	123	MIT ML Device	[MOON]
85	125	MIT ML Device	[MOON]
87	127	any terminal link	[JBP]
89	131	SU/MIT Telnet Gateway	[MRC]
91	133	MIT Dover Spooler	[EBM]
93	135	BBN RCC Accounting	[DT]
95	137	SUPDUP	[15,MRC]
97	141	Datacomputer Status	[8,JZS]
99	143	CADC - NIFTP via UCL	[PLH]
101	145	NPL - NIFTP via UCL	[PLH]
103	147	BNPL - NIFTP via UCL	[PLH]
105	151	CAMBRIDGE - NIFTP via UCL	[PLH]
107	153	HARWELL - NIFTP via UCL	[PLH]
109	155	SWURCC - NIFTP via UCL	[PLH]
111	157	ESSEX - NIFTP via UCL	[PLH]
113	161	RUTHERFORD - NIFTP via UCL	[PLH]
115-129	163-201	Unassigned	[JBP]
131	203	Datacomputer	[8,JZS]

Reserved for Future Use

Decimal	Octal	Description	References
132-223	204-337	Reserved	[JBP]

Port or Socket Numbers

Experimental Functions

Decimal	Octal	Description	References
224-239	340-357	Unassigned	[JBP]
241	361	NCP Measurement	[9,JBP]
243	363	Survey Measurement	[2,AV]
245	365	LINK	[7,RDB2]
247	367	TIPSRV	[RHT]
249-255	371-377	RSEXEC	[51,RHT]

ASSIGNED LINK NUMBERS

The word "link" here refers to a field in the original ARPANET Host/IMP interface leader. The link was originally defined as an 8 bit field. Some time after the ARPANET Host-to-Host (AHHP) protocol was defined and, by now, some time ago the definition of this field was changed to "Message-ID" and the length to 12 bits. The name link now refers to the high order 8 bits of this 12 bit message-id field. The low order 4 bits of the message-id field are to be zero unless specifically specified otherwise for the particular protocol used on that link. The Host/IMP interface is defined in BBN report 1822 [1].

Link Assignments:

Decimal	Octal	Description	References
0	0	AHHP Control Messages	[28,17,JBP]
1	1	Reserved	[JBP]
2-71	2-107	AHHP Regular Messages	[28,17,JBP]
72-150	110-226	Reserved	[JBP]
151	227	CHAOS Protocol	[MOON]
152	230	PARC Universal Protocol	[4,EAT3]
153	231	TIP Status Reporting	[JGH]
154	232	TIP Accounting	[JGH]
155	233	Internet Protocol (regular)	[33,JBP]
156-158	234-236	Internet Protocol (experimental)	[33,JBP]
159-191	237-277	Measurements	[9,VGC]
192-195	300-303	Unassigned	[JBP]
196-255	304-377	Experimental Protocols	[JBP]
224-255	340-377	NVP	[12,17,OC]
248-255	370-377	Network Maintenance	[JGH]

DOCUMENTS

- [1] BBN, "Specifications for the Interconnection of a Host and an IMP", Report 1822, Bolt Beranek and Newman, Cambridge, Massachusetts, May 1978.
- [2] Bhushan, A., "A Report on the Survey Project", RFC 530, NIC 17375, 22 June 1973.
- [3] Bisbey, R., D. Hollingworth, and B. Britt, "Graphics Language (version 2.1)", ISI/TM-80-18, USC/Information Sciences Institute, July 1980.
- [4] Boggs, D., J. Shoch, E. Taft, and R. Metcalfe, "PUP: An Internetwork Architecture", XEROX Palo Alto Research Center, CSL-79-10, July 1979; also in IEEE Transactions on Communication, Volume COM-28, Number 4, April 1980.
- [5] Braden, R., "NETRJS Protocol", RFC 740, NIC 42423, 22 November 1977. Also in [17].
- [6] Bressler, B., "Remote Job Entry Protocol", RFC 407, NIC 12112, 16 October 72. Also in [17].
- [7] Bressler, R., "Inter-Entity Communication -- An Experiment", RFC 441, NIC 13773, 19 January 1973.
- [8] CCA, "Datacomputer Version 5/4 User Manual", Computer Corporation of America, August 1979.
- [9] Cerf, V., "NCP Statistics", RFC 388, NIC 11360, 23 August 1972.
- [10] Clark, D., "Revision of DSP Specification", Local Network Note 9, Laboratory for Computer Science, MIT, 17 June 1977.
- [11] Clements, R., "FTPSRV -- Extensions for Tenex Paged Files", RFC 683, NIC 32251, 3 April 1975. Also in [17].
- [12] Cohen, D., "Specifications for the Network Voice Protocol (NVP)", NSC Note 68, 29 January 1976. Also as USC/Information Sciences Institute RR-75-39, March 1976, and as RFC 741, NIC 42444, 22 November 1977. Also in [17].
- [13] Cohen, D. and J. Postel, "Multiplexing Protocol", IEN 90, USC/Information Sciences Institute, May 1979.

Documents

- [14] COMPASS, "Semi-Annual Technical Report", CADD-7603-0411, Massachusetts Computer Associates, 4 March 1976. Also as, "National Software Works, Status Report No. 1", RADC-TR-76-276, Volume 1, September 1976. And COMPASS, "Second Semi-Annual Report", CADD-7608-1611, Massachusetts Computer Associates, 16 August 1976.
- [15] Crispin, M., "SUPDUP Protocol", RFC 734, NIC 41953, 7 October 1977. Also in [17].
- [16] Crispin, M. and I. Zabala, "DIALNET Protocols", Stanford University Artificial Intelligence Laboratory, July 1978.
- [17] Feinler, E. and J. Postel, eds., "ARPANET Protocol Handbook", NIC 7104, for the Defense Communications Agency by SRI International, Menlo Park, California, Revised January 1978.
- [18] Flood Page, D., "Gateway Monitoring Protocol", IEN 131, February 1980.
- [19] Flood Page, D., "CMCC Performance Measurement Message Formats", IEN 157, September 1980.
- [20] Forgie, J., "ST - A Proposed Internet Stream Protocol", IEN 119, M.I.T. Lincoln Laboratory, September 1979.
- [21] Forsdick, H., and A. McKenzie, "FTP Functional Specification", Bolt Beranek and Newman, Report 4051, August 1979.
- [22] Harrenstien, K., J. Postel, "Time Server", IEN 142, April 1980. Also in [17].
- [23] Harrenstien, K., "Name/Finger", RFC 742, NIC 42758, 30 December 1977. Also in [17].
- [24] Harvey, B., "One More Try on the FTP", RFC 691, NIC 32700, 6 June 1975.
- [25] Haverty, J., "XNET Formats for Internet Protocol Version 4", IEN 158, October 1980.
- [26] McCarthy, J. and L. Earnest, "DIALNET", Stanford University Artificial Intelligence Laboratory, Undated.
- [27] McKenzie, A., "File Transfer Protocol", RFC 454, NIC 14333, 16 February 1973.

Documents

- [28] McKenzie, A., "Host/Host Protocol for the ARPA Network", NIC 8246, January 1972. Also in [17].
- [29] NSW Protocol Committee, "MSG: The Interprocess Communication Facility for the National Software Works", CADD-7612-2411, Massachusetts Computer Associates, BBN 3237, Bolt Beranek and Newman, Revised 24 December 1976.
- [30] O'Brien, M., "A Network Graphical Conferencing System", RAND Corporation, N-1250-ARPA, August 1979.
- [31] Postel, J., "Character Generator Process", RFC 429, NIC 13281, 12 December 1972.
- [32] Postel, J., "Discard Process", RFC 348, NIC 10427, 30 May 1972.
- [33] Postel, J., ed., "Internet Protocol - DARPA Internet Program Protocol Specification", RFC 791, USC/Information Sciences Institute, September 1981.
- [34] Postel, J., ed., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, USC/Information Sciences Institute, September 1981.
- [35] Postel, J., "Echo Process", RFC 347, NIC 10426, 30 May 1972.
- [36] Postel, J., "File Transfer Protocol", RFC 765, IEN 149, June 1980.
- [37] Postel, J., "Internet Message Protocol", RFC 759, IEN 113, USC/Information Sciences Institute, August 1980.
- [38] Postel, J., "Name Server", IEN 116, USC/Information Sciences Institute, August 1979.
- [39] Postel, J., "Official Initial Connection Protocol", NIC 7101, 11 June 1971. Also in [17].
- [40] Postel, J., "Telnet Protocol", RFC 318, NIC 9348, 3 April 1972.
- [41] Postel, J., "Telnet Protocol Specification", RFC 764, IEN 148, June 1980.
- [42] Postel, J., "User Datagram Protocol", RFC 768 USC/Information Sciences Institute, August 1980.

Documents

- [43] Reed, D., "Protocols for the LCS Network", Local Network Note 3, Laboratory for Computer Science, MIT, 29 November 1976.
- [44] Skelton, A., S. Holmgren, and D. Wood, "The MITRE Cablenet Project", IEN 96, April 1979.
- [45] Sluizer, S., and J. Postel, "Mail Transfer Protocol", RFC 780, USC/Information Sciences Institute, May 1981.
- [46] Sproull, R., and E. Thomas, "A Networks Graphics Protocol", NIC 24308, 16 August 1974. Also in [17].
- [47] Sollins, K., "The TFTP Protocol (revision 2)", RFC 783, MIT/LCS, June 1981.
- [48] Strazisar, V., "Gateway Routing: An Implementation Specification", IEN 30, Bolt Berenak and Newman, April 1979.
- [49] Strazisar, V., "How to Build a Gateway", IEN 109, Bolt Berenak and Newman, August 1979.
- [50] The High Level Protocol Group, "A Network Independent File Transfer Protocol", INWG Protocol Note 86, December 1977.
- [51] Thomas, R., "A Resource Sharing Executive for the ARPANET", AFIPS Conference Proceedings, 42:155-163, NCC, 1973.
- [52] Flood Page, D., "A Simple Message Generator", IEN 172, Bolt Berenak and Newman, March 1981.
- [53] Postel, J., "Internet Control Message Protocol - DARPA Internet Program Protocol Specification", RFC 792, USC/Information Sciences Institute, September 1981.
- [54] Postel, J., "Simple Mail Transfer Protocol", RFC 788, USC/Information Sciences Institute, September 1981.
- [55] Littauer, B., "A Host Monitoring Protocol", IEN 197, Bolt Berenak and Newman, September 1981.

PEOPLE

[DCA2]	Don Allen	BBN	Allen@BBND
[CJB]	Chris Bennett	UCL	UKSAT@ISIE
[RB6]	Richard Bisbey	ISI	Bisbey@ISIB
[RT8]	Bob Braden	UCLA	Braden@ISIA
[ROB2]	Robert Bressler	BBN	Bressler@BBNE
[EC5]	Ed Cain	DCEC	cain@EDN-Unix
[VGC]	Vint Cerf	ARPA	Cerf@ISIA
[NC3]	J. Noel Chiappa	MIT	JNC@MIT-XX
[SGC]	Steve Chipman	BBN	Chipman@BBNA
[DDC2]	David Clark	MIT	Clark@MIT-Multics
[DC]	Danny Cohen	ISI	Cohen@ISIB
[MRC]	Mark Crispin	Stanford	Admin.MRC@SU-SCORE
[BD2]	Brian Davies	RSRE	T45@ISIE
[JAKE]	Jake Feinler	SRI	Feinler@SRI-KL
[DFP]	David Flood Page	BBN	DFloodPage@BBNE
[JWF]	Jim Forgie	LL	Forgie@BBNC
[SWG]	Stu Galley	MIT	SWG@MIT-DMS
[GEOF]	Geoff Goodfellow	SRI	Geoff@DARCOM-KA
[KLH]	Ken Harrenstien	MIT	KLH@MIT-AI
[JFH2]	Jack Haverty	BBN	JHaverty@BBN-Unix
[JGH]	Jim Herman	BBN	Herman@BBNE
[PLH]	Peter Higginson	UCL	UKSAT@ISIE
[RH6]	Robert Hinden	BBN	Hinden@BBNE
[CH2]	Charles Hornig	Honeywell	Hornig@MIT-Multics
[EAK]	Earl Killian	LLL	EAK@MIT-MC
[PK]	Peter Kirstein	UCL	Kirstein@ISIA
[DRL]	David Lyons	DEC	Lyons@DEC-2136
[HM]	Hank Magnuski	---	---
[JEM]	Jim Mathis	SRI	Mathis@SRI-KL
[DM11]	Dale McNeill	BBN	DMcNeill@BBNE
[DLM1]	David Mills	COMSAT	Mills@ISIE
[MOON]	David Moon	MIT	Moon@MIT-MC
[EBM]	Eliot Moss	MIT	EBM@MIT-XX
[MO2]	Michael O'Brien	RAND	OBrien@RAND-Unix
[KTP]	Ken Pogran	BBN	Pogran@BBND
[JBP]	Jon Postel	ISI	Postel@ISIF
[JZS]	Joanne Sattely	CCA	JZS@CCA
[APS]	Anita Skelton	MITRE	skelton@MITRE
[KRS]	Karen Sollins	MIT	Sollins@MIT-XX
[VMS]	Virginia Strazisar	BBN	Strazisar@BBNA
[EAT3]	Ed Taft	XEROX	Taft.PA@PARC
[DT]	Dan Tappan	BBN	Tappan@BBNG
[RHT]	Robert Thomas	BBN	Thomas@BBNA
[AV]	Al Vezza	MIT	AV@MIT-XX
[CJW2]	Cliff Weinstein	LL	cjw@LL-11

RFC-794

PRE-EMPTION

September 1981

(427)

PRE-EMPTION

In circuit-switching systems, once a user has a circuit, the communication bandwidth of that circuit is dedicated, even if it is not used. When the system saturates, additional circuit set-up requests are blocked. To allow high precedence users to gain access to circuit resources, systems such as AUTOVON associate a precedence with each telephone instrument. Those instruments with high precedence can pre-empt circuit resources, causing lower precedence users to be cut off.

In message switching systems such as AUTODIN I, incoming traffic is stored on disks (or drums or tape) and processed in order of precedence. If a high precedence message is entered into the system, it is processed and forwarded as quickly as possible. When the high precedence message arrives at the destination message switch, it may pre-empt the use of the output devices on the switch, interrupting the printing of a lower precedence message.

In packet switching systems, there is little or no storage in the transport system so that precedence has little impact on delay for processing a packet. However, when a packet switching system reaches saturation, it rejects offered traffic. Precedence can be used in saturated packet switched systems to sort traffic queued for entry into the system.

In general, precedence is a tool for deciding how to allocate resources when systems are saturated. In circuit switched systems, the resource is circuits; in message switched systems the resource is the message switch processor; and in packet switching the resource is the packet switching system itself.

This capability can be realized in AUTODIN II without adding any new mechanisms to TCP (except to make precedence of incoming connection requests visible to the processes which use TCP). To allow pre-emptive access to a particular terminal, the software (i.e., THP) which supports terminal access to the TAC can be configured so as to always have a LISTEN posted for that terminal, even if the terminal has a connection in operation. For example in the ARPANET TENEX systems, the user TELNET permits a user to have many connections open at one time - the user can switch among them at will. To the extent that this can be done without violating security requirements, one could imagine a multi-connection THP which always leaves a LISTEN pending for incoming connection requests. If a connection is established, the THP can decide, based on its precedence, whether to pre-empt any existing connection and to switch the user to the high precedence one.

If the user is working with several connections of different precedence at the same time, the THP would close or abort the lowest precedence

September 1981

Pre-Emption

connection in favor of the higher precedence pre-empting one. Then the THP would do a new LISTEN on that terminal's port in case a higher precedence connection is attempted.

One of the reasons for suggesting this model is that processes are the users of TCP (in general) and that TCP itself cannot cause processes to be created on behalf of an incoming connection request. Implementations could be realized in which TCPs accept incoming connection requests and, based on the destination port number, create appropriate server processes. In terms of pre-empting access to a remote terminal, however, it seems more sensible to let the process which interfaces the terminal to the system mediate the pre-emption. If the terminal is not connected or is turned off, there is no point in creating a process to serve the incoming high precedence connection request.

For example, suppose a routine FTP is in operation between Host X and Host Y. Host Z decides to do a flash-override FTP to Host X. It opens a high precedence connection via its TCP and the "SYN" goes out to the FTP port on Host X.

FTP always leaves one LISTEN pending to pre-empt lower precedence remote users if it cannot serve one more user (and still keep a LISTEN pending). In this way, the FTP is naturally in a state permitting the high precedence connection request to be properly served, and the FTP can initiate any cleaning up that is needed to deal with the pre-emption.

In general, this strategy permits the processes using TCP to accommodate pre-emption in the context of the applications they support.

A non-pre-emptable process is one that does not have a LISTEN pending while it is serving one (or more) users.

The actions taken to deal with pre-emption of TCP connections will be application-process specific and this strategy of a second (or N+1st) LISTEN is well suited to the situation.

Pre-emption may also be necessary at the site initiating a high precedence connection request. Suppose there is a high precedence user who wants to open an FTP connection request from Host Z to Host X. But all FTP and/or TCP resources are saturated when this user tries to start the user FTP process. In this case, the operating system would have to know about the precedence of the user and would have to locally pre-empt resources on his behalf (e.g., by logging out lower precedence users). This is a system issue, not specific only to TCP. Implementation of pre-emption at the source could vary greatly. Precedence may be associated with a user or with a terminal. The TCP implementation may locally pre-empt resources to serve high precedence users. The operating system may make all pre-emption decisions.

RFC-795

SERVICE MAPPINGS

September 1981

(431)

SERVICE MAPPINGS

This memo describes the relationship between the Internet Protocol (IP) [1] Type of Service and the service parameters of specific networks.

The IP Type of Service has the following fields:

Bits 0-2: Precedence.
Bit 3: 0 = Normal Delay, 1 = Low Delay.
Bits 4: 0 = Normal Throughput, 1 = High Throughput.
Bits 5: 0 = Normal Reliability, 1 = High Reliability.
Bit 6-7: Reserved for Future Use.

0	1	2	3	4	5	6	7
PRECEDENCE			D	T	R	0	0

111 - Network Control
110 - Internetwork Control
101 - CRITIC/ECP
100 - Flash Override
011 - Flash
010 - Immediate
001 - Priority
000 - Routine

The individual networks listed here have very different and specific service choices.

AUTODIN II

The service choices are in two parts: Traffic Acceptance Categories, and Application Type. The Traffic Acceptance Categories can be mapped into and out of the IP TOS precedence reasonably directly. The Application types can be mapped into the remaining IP TOS fields as follows.

TA	DELAY	THROUGHPUT	RELIABILITY
I/A	1	0	0
Q/R	0	0	0
B1	0	1	0
B2	0	1	1
DTR	TA		
---	---		
000	Q/R		
001	Q/R		
010	B1		
011	B2		
100	I/A		
101	I/A		
110	I/A		
111	error		

ARPANET

The service choices are in quite limited. There is one priority bit that can be mapped to the high order bit of the IP TOS precedence. The other choices are to use the regular ("Type 0") messages vs. the uncontrolled ("Type 3") messages, or to use single packet vs. multipacket messages. The mapping of ARPANET parameters into IP TOS parameters can be as follows.

Type	Size	DELAY	THROUGHPUT	RELIABILITY
0	S	1	0	0
0	M	0	0	0
3	S	1	0	0
3	M	not allowed		

DTR	Type	Size
000	0	M
001	0	M
010	0	M
011	0	M
100	3	S
101	0	S
110	3	S
111	error	

PRNET

There is no priority indication. The two choices are to use the station routing vs. point-to-point routing, or to require acknowledgments vs. having no acknowledgments. The mapping of PRNET parameters into IP TOS parameters can be as follows.

Routing	Acks	DELAY	THROUGHPUT	RELIABILITY
ptp	no	1	0	0
ptp	yes	1	0	1
station	no	0	0	0
station	yes	0	0	1

DTR	Routing	Acks
000	station	no
001	station	yes
010	station	no
011	station	yes
100	ptp	no
101	ptp	yes
110	ptp	no
111	ptp	yes

SATNET

There is no priority indication. The four choices are to use the block vs. stream type, to select one of four delay categories, to select one of two holding time strategies, or to request one of three reliability levels. The mapping of SATNET parameters into IP TOS parameters can thus quite complex there being $2 \cdot 4 \cdot 2 \cdot 3 = 48$ distinct possibilities.

References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.

RFC-796

ADDRESS MAPPINGS

September 1981

(437)

Internet Addresses

An internet address is a 32 bit quantity, with several codings as shown below.

																1																	2																	3																
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1																									
0										NETWORK																				Local Address																																				

The second type (or class b) of address has a 14-bit network number and a 16-bit local address.

1 2 3

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
										NETWORK										Local Address											

The third type (or class c) of address has a 21-bit network number and a 8-bit local address.

										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1								
1 1 0										NETWORK										Local Address																			

The local address carries information to address a host in the network identified by the network number. Since each network has a

particular address format and length, the following section describes the mapping between internet local addresses and the actual address format used in the particular network.

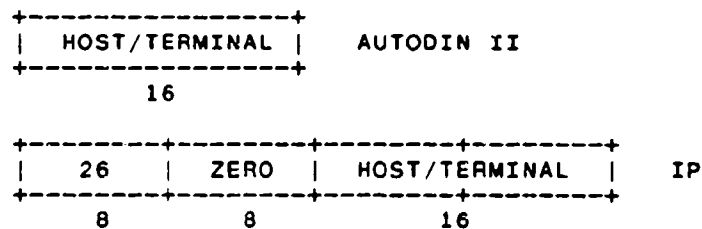
Internet to Local Net Address Mappings

The following transformations are used to convert internet addresses to local net addresses and vice versa:

AUTODIN II

The AUTODIN II has 16 bit subscriber addresses which identify either a host or a terminal. These addresses may be assigned independent of location. The 16 bit AUTODIN II address is located in the 24 bit internet local address as shown below.

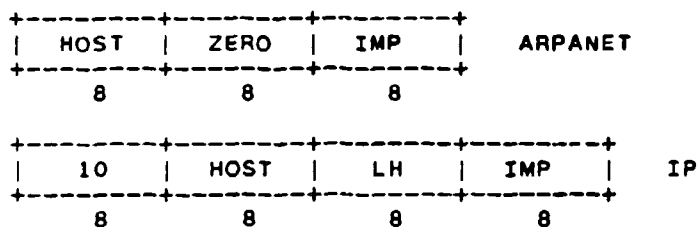
The network number of the AUTODIN II is 26 (Class A).



ARPANET

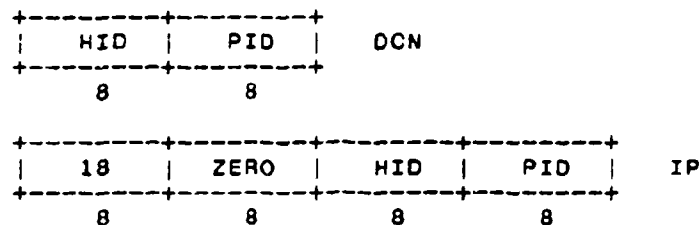
The ARPANET (with 96 bit leaders) has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the ARPANET IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

The network number of the ARPANET is 10 (Class A).

DCNs

The Distributed Computing Networks (DCNs) at COMSAT and UCL use 16 bit addresses divided into an 8 bit host identifier (HID), and an 8 bit process identifier (PID). The format locates these 16 bits in the low order 16 bits of the 24 bit internet address, as shown below.

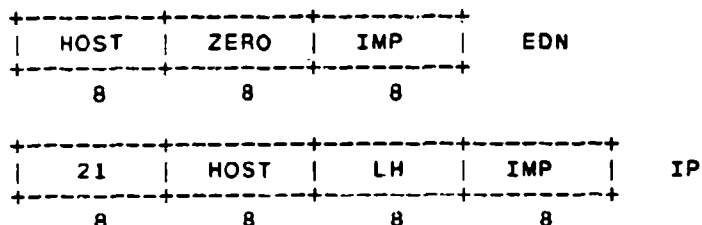
The network number of the COMSAT-DCN is 29 (Class A), and of the UCL-DCN is 30 (Class A).



EDN

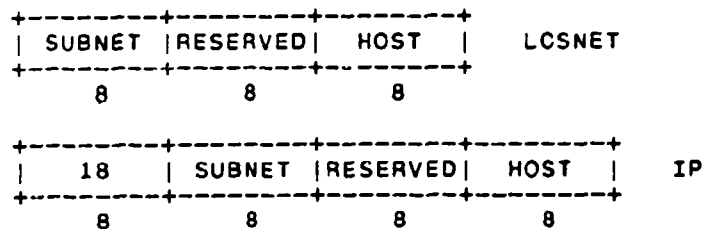
The Experimental Data Network at the Defense Communication Engineering Center (DCEC) uses the same type of addresses as the ARPANET (with 96 bit leaders) and has 24 bit addresses. The 24 bits are assigned to host, logical host, and IMP leader fields as illustrated below. These 24 bit addresses are used directly for the 24 bit local address of the internet address. However, the IMPs do not yet support this form of logical addressing so the logical host field is set to zero in the leader.

The network number of the EDN is 21 (Class A).

LCSNET

The LCS NET at MIT's Laboratory for Computer Science uses 32 bit addresses of several formats. Please see [3] for more details. The most common format locates the low order 24 bits of the 32 bit LCS NET address in the 24 bit internet local address, as shown below.

The network number of the LCS NET is 18 (Class A).

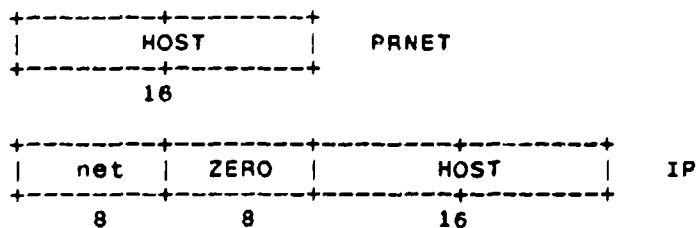


PRNET

The Packet Radio networks use 16 bit addresses. These are independent of location (indeed the hosts may be mobile). The 16 bit PRNET addresses are located in the 24 bit internet local address as shown below.

The network numbers of the PRNETs are:

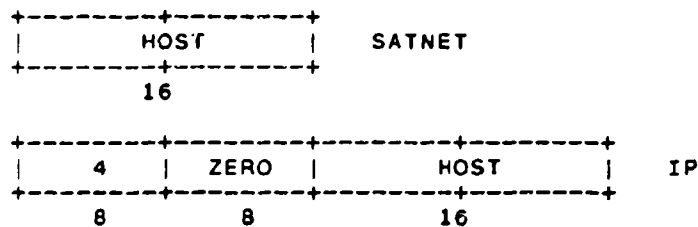
BBN-PR	1 (Class A)
SF-PR-1	2 (Class A)
SILL-PR	5 (Class A)
SF-PR-2	6 (Class A)
BRAGG-PR	9 (Class A)
DC-PR	20 (Class A)



SATNET

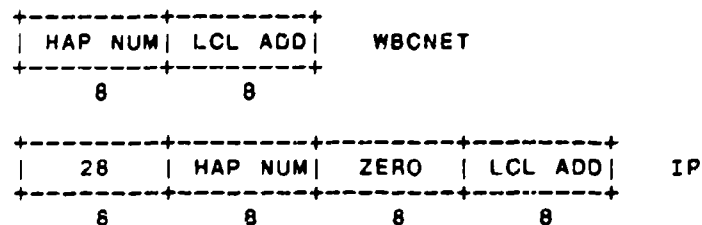
The Atlantic Satellite Packet Network has 16 bit addresses for hosts. These addresses may be assigned independent of location (i.e., ground station). It is also possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit SATNET address is located in the 24 bit internet local address as shown below.

The network number of the SATNET is 4 (Class A).

WBCNET

The Wideband Communication Satellite Packet Network (WBCNET) Host Access Protocol (HAP) has 16 bit addresses for hosts. It is possible to assign several addresses to one physical host, so the addresses are logical addresses. The 16 bit WBCNET address is divided into a HAP Number field and a Local Address field, and is located in the 24 bit internet local address as shown below. Please see [2] for more details.

The network number of the WBCNET is 28 (Class A).



References

- [1] Postel, J. (ed.), "Internet Protocol - DARPA Internet Program Protocol Specification," RFC 791, USC/Information Sciences Institute, September 1981.
- [2] Pershing J., "Addressing Revisited," Bolt Beranek and Newman Inc., W Note 27, May 1981.
- [3] Noel Chiappa, David Clark, David Reed, "LCS Net Address Format," M.I.T. Laboratory for Computer Science Network Implementation, Note No.5, IEN 82, February 1979.

RFC-733

**MAIL HEADER
FORMAT STANDARDS**

21 November 1977

(447)

RFC # 733
NIC # 41952

Obsoletes: RFC #561 (NIC #18516)
RFC #680 (NIC #32116)
RFC #724 (NIC #37435)

STANDARD FOR THE FORMAT OF ARPA NETWORK TEXT MESSAGES⁽¹⁾

21 November 1977

by

David H. Crocker
The Rand Corporation

John J. Vittal
Bolt Beranek and Newman Inc.

Kenneth T. Pogran
Massachusetts Institute of Technology

D. Austin Henderson, Jr.⁽²⁾
Bolt Beranek and Newman Inc.

(1) This work was supported by the Defense Advanced Research Projects Agency of the Department of Defense, under contract Nos. N00014-75-C-0661, MDA903-76-C-0212, and DAHC15-73-C0181.

(2) The authors' postal addresses are: D. Crocker, The Rand Corporation, Information Sciences Dept., 1700 Main St., Santa Monica, California 90406; J. Vittal & D. A. Henderson, Bolt Beranek & Newman, 50 Moulton St., Cambridge, Massachusetts 02138; and K. Pogran, MIT Laboratory for Computer Science, 545 Technology Square, Cambridge, Massachusetts 02139. The authors' ARPANET addresses are: DCrocker at Rand-Unix, Vittal at BBN-TenexD, Pogran at MIT-Multics, and Henderson at BBN-TenexD.

(449)

PREFACE

ARPA's Committee on Computer-Aided Human Communication (CAHCOM) wishes to promulgate a standard for the format of ARPA Network text message (mail) headers which will reasonably meet the needs of the various message service subsystems on the Network today. The authors of this document constitute the CAHCOM subcommittee charged with the task of developing this new standard.

Essentially, we specify a revision to ARPANET Request for Comments (RFC) 561, "Standardizing Network Mail Headers", and RFC 680, "Message Transmission Protocol". This revision removes and compacts portions of the previous syntax and adds several features to network address specification. In particular, we focus on people and not mailboxes as recipients and allow reference to stored address lists. We expect this syntax to provide sufficient capabilities to meet most users' immediate needs and, therefore, give developers enough breathing room to produce a new mail transmission protocol "properly". We believe that there is enough of a consensus in the Network community in favor of such a standard syntax to make possible its adoption at this time. An earlier draft of this specification was published as RFC #724, "Proposed Official Standard for the Format of ARPA Network Messages" and contained extensive discussion of the background and issues in ARPANET mail standards.

This specification was developed over the course of one year, using the ARPANET mail environment, itself, to provide an on-going forum for discussing the capabilities to be included. More than twenty individuals, from across the country, participated in this discussion and we would like to acknowledge their considerable efforts. The syntax of the standard was originally specified in the Backus-Naur Form (BNF) meta-language. Ken L. Harrenstien, of SRI International, was responsible for re-coding the BNF into an augmented BNF which compacts the specification and allows increased comprehensibility.

CONTENTS

PREFACE.....	111
Section	
I. INTRODUCTION.....	1
II. FRAMEWORK.....	2
III. SYNTAX.....	4
A. Notational Conventions.....	4
B. Lexical Analysis of Messages.....	5
C. General Syntax of Messages.....	13
D. Syntax of General Addressee Items.....	15
E. Supporting Constructs.....	16
IV. SEMANTICS.....	17
A. Address Fields.....	17
B. Reference Specification Fields.....	22
C. Other Fields and Syntactic Items.....	23
D. Dates and Times.....	24
V. EXAMPLES.....	25
A. Addressees.....	25
B. Address Lists.....	26
C. Originator Items.....	26
D. Complete Headers.....	28
Appendix	
A. ALPHABETICAL LISTING OF SYNTAX RULES.....	31
B. SIMPLE PARSING.....	35
BIBLIOGRAPHY.....	37

I. INTRODUCTION

This standard specifies a syntax for text messages which are passed between computer users within the framework of "electronic mail". The standard supersedes the informal standards specified in ARPANET Request for Comments numbers 561, "Standardizing Network Mail Headers", and 680, "Message Transmission Protocol". In this document, a general framework is first described; the formal syntax is then specified, followed by a discussion of the semantics. Finally, a number of examples are given.

This specification is intended strictly as a definition of what is to be passed between hosts on the ARPANET. It is NOT intended to dictate either features which systems on the Network are expected to support, or user interfaces to message creating or reading programs.

A distinction should be made between what the specification REQUIRES and what it ALLOWS. Messages can be made complex and rich with formally-structured components of information or can be kept small and simple, with a minimum of such information. Also, the standard simplifies the interpretation of differing visual formats in messages. These simplifications facilitate the formal specification and indicate what the OFFICIAL semantics are for messages. Only the visual aspect of a message is affected and not the interpretation of information within it. Implementors may choose to retain such visual distinctions.

II. FRAMEWORK

Since there are many message systems which exist outside the ARPANET environment, as well as those within it, it may be useful to consider the general framework, and resulting capabilities and limitations, provided by this standard.

Messages are expected to consist of lines of text. No special provisions are made, at this time, for encoding drawings, facsimile, speech, or structured text.

No significant consideration has been given to questions of data compression or transmission/storage efficiency. The standard, in fact, tends to be very free with the number of bits consumed. For example, field names are specified as free text, rather than special terse codes.

A general "memo" framework is used. That is, a message consists of some information, in a rigid format, followed by the main part of the message, which is text and whose format is not specified in this document. The syntax of several fields of the rigidly-formatted ("header") section is defined in this specification; some of the header fields must be included in all messages. The syntax which distinguishes between headers is specified separately from the internal syntax for particular headers. This separation is intended to allow extremely simple parsers to operate on the overall structure of messages, without concern for the detailed structure of individual headers. Appendix B is provided to facilitate construction of these simple parsers. In addition to the fields specified in this document, it is expected that other fields will gain common use. User-defined header fields allow systems to extend their functionality while maintaining a uniform framework. The approach is similar to that of the TELNET protocol, in that a basic standard is defined which includes a mechanism for (optionally) extending itself. As necessary, the authors of this document will regulate the publishing of specifications for these "extension-fields", through the same mechanisms used to publish this document.

Such a framework severely constrains document tone and appearance and is primarily useful for most intra-organization communications and relatively structured inter-organization communication. A more robust environment might allow for multi-font, multi-color, multi-dimension encoding of information. A less robust environment, as is present in most single-machine message systems, would more severely constrain the ability to add fields and the decision to include specific fields. In contrast to paper-based communication, it is interesting to note that the

Standard for the Format of Text Messages
II. Framework

3

RECEIVER of a message can exercise an extraordinary amount of control over the message's appearance. The amount of actual control available to message receivers is contingent upon the capabilities of their individual message systems.

III. SYNTAX

This syntax is given in five parts. The first part describes the notation used in the specification. The second part describes the base-level lexical analyzers which feed the higher-level parser described in the succeeding sections. The third part gives a general syntax for messages and standard header fields; and the fourth part specifies the syntax of addresses. A final part specifies some general syntax which supports the other sections.

A. NOTATIONAL CONVENTIONS

These specifications are made in an augmented Backus-Naur Form (BNF). Differences from standard BNF involve the naming of rules, the indication of repetition and of "local" alternatives.

1. Rule naming

Angle brackets (" $<$ ", " $>$ ") are not used, in general. The name of a rule is simply the name itself, rather than " $<name>$ ". Quotation-marks enclose literal text (which may be upper and/or lower case). Certain basic rules are in uppercase, such as SPACE, TAB, CRLF, DIGIT, ALPHA, etc. Angle brackets are used in rule definitions, and in the rest of this document, whenever their presence will facilitate discerning the use of rule names.

2. Parentheses: Local alternatives

Elements enclosed in parentheses are treated as a single element. Thus, " $(elem (foo / bar) elem)$ " allows " $(elem foo elem)$ " and " $(elem bar elem)$ ".

3. * construct: Repetition

The character "*" preceding an element indicates repetition. The full form is:

$<l>* <m> element$

indicating at least $<l>$ and at most $<m>$ occurrences of element. Default values are 0 and infinity so that " $*(element)$ " allows any number, including zero; " $1*element$ " requires at least one; and " $1*2element$ " allows one or two.

III. Syntax

A. Notational Conventions

4. <number>element

"<n>(element)" is equivalent to "<n>*<n>(element)": that is, exactly <n> occurrences of (element). Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

5. # construct: Lists

A construct "#" is defined, similar to "*", as follows:

<l>#<m>element

indicating at least <l> and at most <m> elements, each separated by one or more commas (","). This makes the usual form of lists very easy; a rule such as '(element *("," element))' can be shown as "l#element". Wherever this construct is used, null elements are allowed, but do not contribute to the count of elements present. That is, "(element)..(element)" is permitted, but counts as only two elements. Therefore, where at least one element is required, at least one non-null element must be present.

6. [optional]

Square brackets enclose optional elements; "[foo bar]" is equivalent to "*1(foo bar)".

7. ; Comments

A semi-colon, set off some distance to the right of rule text, starts a comment which continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

B. LEXICAL ANALYSIS OF MESSAGES

1. General Description

A message consists of headers and, optionally, a body (i.e. a series of text lines). The text part is just a sequence of lines containing ASCII characters; it is separated from the headers by a null line (i.e., a line with nothing preceding the CRLF).

B. Lexical Analysis

Each header item can be viewed as a single, logical line of ASCII characters. For convenience, the field-body portion of this conceptual entity can be split into a multiple-line representation (i.e., "folded"). The general rule is that wherever there can be linear-white-spaces (NOT simply LWSP-chars), a CRLF immediately followed by AT LEAST one LWSP-char can instead be inserted. (However, a header's name and the following colon (":"), which occur at the beginning of the header item, may NOT be folded onto multiple lines.) Thus, the single line

can be represented as

and

and

The process of moving from this folded multiple-line representation of a header field to its single line representation will be called "unfolding". Unfolding is accomplished by regarding CRLF immediately followed by a LWSP-char as equivalent to the LWSP-char.

Once header fields have been unfolded, they may be viewed as being composed of a field-name followed by a colon (":") followed by a field-body. The field-name must be composed of printable ASCII characters (i.e., characters which have values between 33. and 126., decimal, except colon) and LWSP-chars. The field-body may be composed of any ASCII characters (other than an unquoted CRLF, which has been removed by unfolding).

(460)

III. Syntax

B. Lexical Analysis

addresses. Other fields, such as "Subject" and "Comments", are regarded simply as strings of text.

NOTE: Field-names, unstructured field bodies and structured field bodies each are scanned by their own, INDEPENDENT "lexical" analyzer.

c. Field-names

To aid in the creation and reading of field-names, the free insertion of LWSP-chars is allowed in reasonable places.

Rather than obscuring the syntax specification for field-name with the explicit syntax for these LWSP-chars, the existence of a "lexical" analyzer is assumed. The analyzer interprets the text which comprises the field-name as a sequence of field-name atoms (fnatoms) separated by LWSP-chars

Note that ONLY LWSP-chars may occur between the fnatoms of a field-name and that CRLFs may NOT. In addition, comments are NOT lexically recognized, as such, but parenthesized strings are legal as part of field-names. These constraints are different from what is permissible within structured field bodies. In particular, this means that header field-names must wholly occur on the FIRST line of a folded header item and may NOT be split across two or more lines.

d. Unstructured field bodies

For some fields, such as "Subject" and "Comments", no structuring is assumed; and they are treated simply as texts, like those in the message body. Rules of folding apply to these fields, so that such field bodies which occupy several lines must therefore have the second and successive lines indented by at least one LWSP-char.

e. Structured field bodies

To aid in the creation and reading of structured fields, the free insertion of linear-white-space (which permits folding by inclusion of CRLFs) is allowed in reasonable places. Rather than obscuring the syntax specifications for these structured fields with explicit syntax for this linear-white-space, the existence of another "lexical" analyzer is assumed. This analyzer does not apply for field bodies which are simply unstructured strings of text, as described above. It provides an interpretation of the unfolded text comprising the body of the field as a sequence of lexical symbols. These symbols are:

- individual special characters
- quoted-strings

(461)

III. Syntax

B. Lexical Analysis

- comments
- atoms

The first three of these symbols are self-delimiting. Atoms are not; they therefore are delimited by the self-delimiting symbols and by linear-white-space. For the purposes of re-generating sequences of atoms and quoted-strings, exactly one SPACE is assumed to exist and should be used between them. (Also, in Section III.8.3.a, note the rules concerning treatment of multiple contiguous LWSP-chars.)

So, for example, the folded body of an address field

```
":syemail"@ Some-Host,
Muhammed(I am the greatest)Ali at(the)WBA
```

is analyzed into the following lexical symbols and types:

":syemail"	quoted string
@	special
Some-Host	atom
,	special
Muhammed	atom
(I am the greatest)	comment
Ali	atom
at	atom
(the)	comment
WBA	atom

The cononical representations for the data in these addresses are the following strings (note that there is exactly one SPACE between words):

```
:syemail at Some-Host
```

and

```
Muhammed Ali at WBA
```

2. Formal Definitions

The first four rules, below, indicate a meta-syntax for fields, without regard to their particular type or internal syntax. The remaining rules define basic syntactic structures which are used by the rules in Sections III.C, III.D, and III.E.

```
field      = field-name ":" [ field-body ] CRLF
```

```
field-name = fnatom *( LWSP-char [fnatom] )
```

(462)

III. Syntax

B. Lexical Analysis

fnatom = 1*<any CHAR, excluding CTLs, SPACE, and ":">

field-body = field-body-contents
[CRLF LWSP-char field-body]

field-body-contents = <the TELNET ASCII characters making up the field-body, as defined in the following sections, and consisting of combinations of atom, quoted-string, and specials tokens, or else consisting of texts>

CHAR	=	<any TELNET ASCII character>	:	(Octal, Decimal.)
ALPHA	=	<any TELNET ASCII alphabetic character>	:	(0-177, 0.-127.)
			:	(101-132, 65.- 90.)
			:	(141-172, 97.-122.)
DIGIT	=	<any TELNET ASCII digit>	:	(60- 71, 48.- 57.)
CTL	=	<any TELNET ASCII control character and DEL>	:	(0- 37, 0.- 31.)
			:	(177, 127.)
CR	=	<TELNET ASCII carriage return>	:	(15, 13.)
LF	=	<TELNET ASCII linefeed>	:	(12, 10.)
SPACE	=	<TELNET ASCII space>	:	(40, 32.)
HTAB	=	<TELNET ASCII horizontal-tab>	:	(11, 9.)
<">	=	<TELNET ASCII quote mark>	:	(42, 34.)
CRLF	=	CR LF			

LWSP-char = SPACE / HTAB ; semantics = SPACE
linear-white-space = 1*([CRLF] LWSP-char) ; semantics = SPACE
; CRLF => folding

specials = "(" / ")" / "<" / ">" / "@" ; To use in a word.
/ "." / ";" / ":" / "\" / "<" ; word must be a
; quoted-string.

delimiters = specials / comment / linear-white-space

text = <any CHAR, including bare ; => atoms, specials,
CR and/or bare LF, but NOT ; comments and
including CRLF> ; quoted-strings are
; NOT interpreted.

atom = 1*<any CHAR except specials and CTLs>

quoted-string = <"> *(qtext/quoted-pair) <">; Any number of qtext
; chars or any
; quoted char.

qtext = <any CHAR excepting <"> ; => may be folded
and CR, and including
linear-white-space>

III. Syntax

B. Lexical Analysis

comment = "(" *(ctext / comment / quoted-pair) ")"
ctext = <any CHAR excluding "(", ")", and CR, and including
linear-white-space>

quoted-pair = "\" CHAR

3. Clarifications

a. "White space"

Remember that in field-names and structured field bodies, MULTIPLE LINEAR WHITE SPACE TELNET ASCII CHARACTERS (namely HTABs and SPACEs) ARE TREATED AS SINGLE SPACES AND MAY FREELY SURROUND ANY SYMBOL. In all header fields, the only place in which at least one space is REQUIRED is at the beginning of continuation lines in a folded field. When passing text to processes which do not interpret text according to this standard (e.g., ARPANET FTP mail servers), then exactly one SPACE should be used in place of arbitrary linear-white-space and comment sequences.

WHEREVER A MEMBER OF THE LIST OF <DELIMITER>S IS ALLOWED, LWSP-CHARS MAY ALSO OCCUR BEFORE AND/OR AFTER IT.

Writers of mail-sending (i.e. header generating) programs should realize that there is no Network-wide definition of the effect of horizontal-tab TELNET ASCII characters on the appearance of text at another Network host; therefore, the use of tabs in message headers, though permitted, is discouraged.

Note that during transmissions across the ARPANET using TELNET NVT connections, data must conform to TELNET NVT conventions (e.g., CR must be followed by either LF, making a CRLF, or <null>, if the CR is to stand alone).

b. Comments

Comments are detected as such only within field-bodies of structured fields. A comment is a set of TELNET ASCII characters, which is not within a quoted-string and which is enclosed in matching parentheses; parentheses nest, so that if an unquoted left parenthesis occurs in a comment string, there must also be a matching right parenthesis. When a comment is used to act as the delimiter between a sequence of two lexical symbols, such as two atoms, it is lexically equivalent with one SPACE, for the purposes of regenerating the sequence, such as when passing the sequence onto an FTP mail server.

III. Syntax

B. Lexical Analysis

In particular comments are NOT passed to the FTP server, as part of a MAIL or MLFL command, since comments are not part of the "formal" address.

If a comment is to be "folded" onto multiple lines, then the syntax for folding must be adhered to. (See items III.B.1.a, above, and III.B.3.f, below.) Note that the official semantics therefore do not "see" any unquoted CRLFs which are in comments, although particular parsing programs may wish to note their presence. For these programs, it would be reasonable to interpret a "CRLF LWSP-char" as being a CRLF which is part of the comment; i.e., the CRLF is kept and the LWSP-char is discarded. Quoted CRLFs (i.e., a backslash followed by a CR followed by a LF) still must be followed by at least one LWSP-char.

c. Delimiting and quoting characters

The quote character (backslash) and characters which delimit syntactic units are not, generally, to be taken as data which are part of the delimited or quoted unit(s). The one exception is SPACE. In particular, the quotation-marks which define a quoted-string, the parentheses which define a comment and the backslash which quotes a following character are NOT part of the quoted-string, comment or quoted character. A quotation-mark which is to be part of a quoted-string, a parenthesis which is to be part of a comment and a backslash which is to be part of either must each be preceded by the quote-character backslash ("\"). Note that the syntax allows any character to be quoted within a quoted-string or comment; however only certain characters MUST be quoted to be included as data. These characters are those which are not part of the alternate text group (i.e., ctext or qtext).

A single SPACE is assumed to exist between contiguous words in a phrase, and this interpretation is independent of the actual number of LWSP-chars which the creator places between the words. To include more than one SPACE, the creator must make the LWSP-chars be part of a quoted-string.

Quotation marks which delimit a quoted string and backslashes which quote the following character should NOT accompany the quoted-string when the string is used with processes that do not interpret data according to this specification (e.g., ARPANET FTP mail servers).

III. Syntax

B. Lexical Analysis

d. Quoted-strings

Where permitted (i.e., in words in structured fields) quoted-strings are treated as a single symbol (i.e. equivalent to an atom, syntactically). If a quoted-string is to be "folded" onto multiple lines, then the syntax for folding must be adhered to. (See items III.B.1.a, above, and III.B.3.f, below.) Note that the official semantics therefore do not "see" any bare CRLFs which are in quoted-strings, although particular parsing programs may wish to note their presence. For these programs, it would be reasonable to interpret a "CRLF LWSP-char" as being a CRLF which is part of the quoted-string; i.e., the CRLF is kept and the LWSP-char is discarded. Quoted CRLFs (i.e., a backslash followed by a CR followed by a LF) are also subject to rules of folding, but the presence of the quoting character (backslash) explicitly indicates that the CRLF is data to the quoted string. Stripping off the first following LWSP-char is also appropriate when parsing quoted CRLFs.

e. Bracketing characters

There are three types of brackets which must be well nested:

- o Parentheses are used to indicate comments.
- o Angle brackets (" $<$ " and " $>$ ") are generally used to indicate the presence of at least one machine-usable code (e.g., delimiting mailboxes).
- o Colon/semi-colon (" $:$ " and " $;$ ") are used in address specifications to indicate that the included list of addresses are to be treated as a group.

f. Case independence of certain specials atoms

Certain atoms, which are represented in the syntax as literal alphabetic strings, can be represented in any combination of upper and lower case. These are:

- field-name,
- "Include", "Postal" and equivalent atoms in a " $:$ "<atom>" address specification,
- "at", in a host-indicator,
- node,
- day-of-week,
- month, and
- zones.

When matching an atom against one of these literals, case is to be ignored. For example, the field-names "From", "FROM",

III. Syntax

B. Lexical Analysis

"from", and even "From" should all be treated identically. However, the case shown in this specification is suggested for message-creating processes. Note that, at the level of this specification, case IS relevant to other words and texts. Also see Section IV.A.1.f, below.

g. Folding long lines

Each header item (field of the message) may be represented on exactly one line consisting of the name of the field and its body; this is what the parser sees. For readability, it is recommended that the field-body portion of long header items be "folded" onto multiple lines of the actual header. "Long" is commonly interpreted to mean greater than 65 or 72 characters. The former length is recommended as a limit, but it is not imposed by this standard.

h. Backspace characters

Backspace TELNET ASCII characters (ASCII BS, decimal 8.) may be included in texts and quoted-strings to effect overstriking; however, any use of backspaces which effects an overstrike to the left of the beginning of the text or quoted-string is prohibited.

C. GENERAL SYNTAX OF MESSAGES:

NOTE: Due to an artifact of the notational conventions, the syntax indicates that, when present, "Date", "From", "Sender", and "Reply-To" fields must be in a particular order. These header items must be unique (occur exactly once). However header fields, in fact, are NOT required to occur in any particular order, except that the message body must occur AFTER the headers. For readability and ease of parsing by simple systems, it is recommended that headers be sent in the order "Date", "From", "Subject", "Sender", "To", "cc", etc. This specification permits multiple occurrences of most optional-fields. However, their interpretation is not specified here, and their use is strongly discouraged.

The following syntax for the bodies of various fields should be thought of as describing each field body as a single long string (or line). The section on Lexical Analysis (section II.8) indicates how such long strings can be represented on more than one line in the actual transmitted message.

III. Syntax

C. Messages

```

message      = fields *( CRLF *text )      ; Everything after
                                              ; first null line
                                              ; is message body

fields       = date-field                  ; Creation time-stamp
              originator-fields            ; & author id's are
              *optional-field              ; required; others
                                              ; are all optional

originator-fields =
    ( "From"      ":" mailbox      ; Single author
      [ "Reply-To" ":" #address ] )
  / ( "From"      ":" 1#address    ; Multiple authors &
      "Sender"    ":" mailbox     ; may have non-mach-
      [ "Reply-To" ":" #address ] ); line addresses

date-field   = "Date"      ":" date-time

optional-field =
    "To"      ":" #address
  / "cc"      ":" #address
  / "bcc"     ":" #address      ; Blind carbon
  / "Subject" ":" *text
  / "Comments" ":" *text
  / "Message-ID" ":" mach-id    ; Only one allowed
  / "In-Reply-To" ":" #(phrase / mach-id)
  / "References" ":" #(phrase / mach-id)
  / "Keywords"  ":" #phrase
  / extension-field            ; To be defined in
                                ; supplemental
                                ; specifications
  / user-defined-field         ; Must have unique
                                ; field-name & may
                                ; be pre-empted

extension-field = <Any field which is defined in a document
                  published as a formal extension to this
                  specification>

user-defined-field = <Any field which has not been defined in
                    this specification or published as an extension to
                    this specification; names for such fields must be
                    unique and may be preempted by published
                    extensions>

```

III. Syntax

D. Addressee Items

D. SYNTAX OF GENERAL ADDRESSEE ITEMS

```

address      = host-phrase
               / ( [phrase] "<" #address ">" )
               / ( [phrase] ":" #address ":" )
               / quoted-string
               / ( ":" ( "Include"
                       / "Postal"
                       / atom )
                 ":" address )
               ; Machine mailbox
               ; Individual / List
               ; Group
               ; Arbitrary text
               ; File, w/ addr list
               ; (U.S.) Postal addr
               ; Extended data type

mailbox      = host-phrase / (phrase mach-id)

mach-id      = "<" host-phrase ">"
               ; Contents must never
               ; be modified!

```

E. SUPPORTING CONSTRUCTS

```

host-phrase  = phrase host-indicator
               ; Basic address

host-indicator = 1*( "at" / "@" ) node
               ; Right-most node is
               ; at top of network
               ; hierarchy; left-
               ; most must be host

node         = word / 1*DIGIT
               ; Official host or
               ; network name or
               ; decimal address

date-time    = [ day-of-week "." ] date time

day-of-week  = "Monday" / "Mon" / "Tuesday" / "Tue"
               / "Wednesday" / "Wed" / "Thursday" / "Thu"
               / "Friday" / "Fri" / "Saturday" / "Sat"
               / "Sunday" / "Sun"

date         = 1*2DIGIT ["-"] month
               ["-"] (2DIGIT / 4DIGIT)
               ; day month year
               ; e.g. 20 Aug [19]77

month        = "January" / "Jan" / "February" / "Feb"
               / "March" / "Mar" / "April" / "Apr"
               / "May" / "June" / "Jun"
               / "July" / "Jul" / "August" / "Aug"
               / "September" / "Sep" / "October" / "Oct"
               / "November" / "Nov" / "December" / "Dec"

```

Standard for the Format of Text Messages
 III. Syntax
 E. Supporting Constructs

18

time	=	hour zone	:	ANSI and Military
			:	(seconds optional)
hour	=	2DIGIT [":"] 2DIGIT [[":"] 2DIGIT]	:	0000[00] - 2359[59]
zone	=	(["-"] ("GMT"	:	Relative to GMT:
		/ "NST" /	:	North American
		/ "AST" / "ADT"	:	Newfoundland: -3:30
		/ "EST" / "EDT"	:	Atlantic: - 4/ - 3
		/ "CST" / "CDT"	:	Eastern: - 5/ - 4
		/ "MST" / "MDT"	:	Central: - 6/ - 5
		/ "PST" / "PDT"	:	Mountain: - 7/ - 6
		/ "YST" / "YDT"	:	Pacific: - 8/ - 7
		/ "HST" / "HDT"	:	Yukon: - 9/ - 8
		/ "BST" / "BDT"	:	Haw/Ala -10/ - 9
		1ALPHA))	:	Bering: -11/ -10
		/ (("+" / "-") 4DIGIT)	:	Military: Z = GMT;
			:	A: -1; (J not used)
			:	M: -12; N: +1; Y: +12
			:	Local differential
			:	hours/min. (HHMM)
phrase	=	1*word	:	Sequence of words.
			:	Separation seman-
			:	tically = SPACE
word	=	atom / quoted-string		

IV. Semantics

A. Address Fields

IV. SEMANTICS

A. ADDRESS FIELDS

1. General

- a. The phrase part of a host-phrase in an address specification (i.e., the host's name for the mailbox) is understood to be whatever the receiving FTP Server allows (for example, TENEX systems do not now understand addresses of the form "P. D. Q. Bach", but another system might).

Note that a mailbox is a conceptual entity which does not necessarily pertain to file storage. For example, some sites may choose to print mail on their line printer and deliver the output to the addressee's desk.

An individual may have several mailboxes and a group of individuals may wish to receive mail as a single unit (i.e., a distribution list). The second and third alternatives of an address list (#address) allow naming a collection of subordinate addresses list(s). Recipient mailboxes are specified within the bracketed part ("<" - ">" or ";" - ";") of such named lists. The use of angle-brackets ("<", ">") is intended for the cases of individuals with multiple mailboxes and of special mailbox lists; it is not expected to be nested more than one level, although the specification allows such nesting. The use of colon/semi-colon (";", ";") is intended for the case of groups. Groups can be expected to nest (i.e., to contain subgroups). For both individuals and groups, a copy of the transmitted message is to be sent to EACH mailbox listed. For the case of a special list, treatment of addresses is defined in the relevant subsections of this section.

- b. The inclusion of bare quoted-strings as addresses (i.e., the fourth address-form alternative) is allowed as a syntactic convenience, but no semantics are defined for their use. However, it is reasonable, when replicating an address list, to replicate ALL of its members, including quoted-strings.
- c. ":Include:" specifications are used to refer to one or more locations containing stored address lists (#address). If more than one location is referenced, the address part of the Include phrase must be a list (#address) surrounded by angle-brackets, as per the "Individual / List" alternative of <address>. Constituent addresses must resolve to a host-

IV. Semantics

A. Address Fields

phrase; only they have any meaning within this construct. The phrase part of indicated host-phrases should contain text which the referenced host can resolve to a file. This standard is not a protocol and so does not prescribe HOW data is to be retrieved from the file. However, the following requirements are made:

- o The file must be accessible through the local operating system interface (if it exists), given adequate user access rights; and
- o If a host has an FTP server and a user is able to retrieve any files from the host using that server, then the file must be accessible through FTP, using DEFAULT transfer settings, given adequate user access rights.

It is intended that this mechanism allow programs to retrieve such lists automatically.

The interpretation of such a file reference follows. This is not intended to imply any particular implementation scheme, but is presented to aid in understanding the notion of including file contents in address lists:

- o Elements of the address list part are alternates and the contents of ONLY ONE of them are to be included in the resultant address list.
- o The contents of the file indicated by a member host-phrase are treated as an address list and are inserted as an address list (#address) in the position of the path item in the syntax. That is, the TELNET ASCII characters specifying the entire Include <address> is replaced by the contents of one of the files to which the host-phrase(s) of the address list (#address) refers. Therefore, the contents of each file, indicated by an Include address, must be syntactically self-contained and must adhere to the full syntax prescribed herein for an address list.
- d. ":Postal:" specifications are used to indicate (U.S.) postal addresses, but can be treated the same as quoted-string addresses. To reference a list of postal addresses, the list must conform to the "Individual / List" alternative of <address>. The ":Include:" alternative also is valid.
- e. The "::" atom ":" syntax is intended as a general mechanism for indicating specially data-typed addresses. As with extension-fields, the authors of this document will regulate

IV. Semantics

A. Address Fields

the publishing of specifications for these extended data-types. In the absence of defined semantics, any occurrence of an address in this form may be treated as a quoted-string address.

- f. A node name must be THE official name of a network or a host, or else a decimal number indicating the Network address for that network or host, at the time the message is created. The USE OF NUMBERS IS STRONGLY DISCOURAGED and is permitted only due to the occasional necessity of bypassing local name tables. For the ARPANET, official names are maintained by the Network Information Center at SRI International, Menlo Park, California.

Whenever a message might be transmitted or migrate to a host on another network, full hierarchical addresses must be specified. These are indicated as a series of words, separated by at-sign or "at" indications. The communication environment is assumed to consist of a collection of networks organized as independent "trees" except for connections between the root nodes. That is, only the roots can act as gateways between these independent networks. While other actual connections may exist, it is believed that presuming this type of organization will provide a reliable method for describing VALID, if not EFFICIENT, paths between hosts. A typical full mailbox specification might therefore look like:

Friendly User @ hosta @ local-net1 @ major-netq

In the simplest case, a mail-sending host should transmit the message to the node which is mentioned last (farthest to the right), strip off that node reference from the specification, and then pass the remaining host-phrase to the recipient host (in the ARPANET, its FTP server) for it to process. This treats the remaining portion of the host-indicator merely as the terminating part of the phrase.

NOTE: When passing any portion of a host-indicator onto a process which does not interpret data according to this standard (e.g., ARPANET FTP servers), "@" must be used and not "at" and it must not be preceded or followed by any LWSP-chars. Using the above example, the following string would be passed to the major-netq gateway:

Friendly User@hosta@local-net1

When the sending host has more knowledge of the network environment, then it should send the message along a more efficient path, making appropriate changes to the form of the host-phrase which it gives to the recipient host.

IV. Semantics

A. Address Fields

To use the above specification as an example: If a sending hostb also were part of local-net1, then it could send the message directly to hosta and would give only the phrase "Friendly User" to hosta's mail-receiving program. If hostb were part of local-net2, along with hostc, and happened to know that hosta and hostc were part of another local-net, then hostb could send the message to hostc to the address "Friendly User@hosta".

The phrase in a host-phrase is intended to be meaningful only to the indicated receiving host. To all other hosts, the phrase is to be treated as an uninterpreted string. No case transformations should be (automatically) performed on the phrase. The phrase is passed to the local host's mail sending program; it is the responsibility of the destination host's mail receiving (distribution) program to perform case mapping on this phrase, if required, to deliver the mail.

2. Originator Fields

WARNING: The standard allows only a subset of the combinations possible with the From, Sender, and Reply-To fields. The limitation is intentional.

a. From

This field contains the identity of the person(s) who wished this message to be sent. The message-creation process should default this field to be a single machine address, indicating the AGENT (person or process) entering the message. If this is NOT done, the "Sender" field MUST be present; if this IS done, the "Sender" field is optional.

b. Sender

This field contains the identity of the AGENT (person or process) who sends the message. It is intended for use when the sender is not the author of the message, or to indicate who among a group of authors actually sent the message. If the contents of the "Sender" field would be completely redundant with the "From" field, then the "Sender" field need not be present and its use is discouraged (though still legal); in particular, the "Sender" field MUST be present if it is NOT the same as the "From" Field.

The Sender host-phrase includes a phrase which must correspond to a specific agent (i.e., a human user or a computer program) rather than a standard address. This indicates the expectation that the field will identify the single AGENT (person or process) responsible for sending the

IV. Semantics

A. Address Fields

mail and not simply include the name of a mailbox from which the mail was sent. For example in the case of a shared login name, the name, by itself, would not be adequate. The phrase part of the host-phrase, which refers to this agent, is expected to be a computer system term, and not (for example) a generalized person reference which can be used outside the network text message context.

Since the critical function served by the "Sender" field is the identification of the agent responsible for sending mail and since computer programs cannot be held accountable for their behavior, is strongly recommended that when a computer program generates a message, the HUMAN who is responsible for that program be referenced as part of the "Sender" field host-phrase.

c. Reply-To

This field provides a general mechanism for indicating any mailbox(es) to which responses are to be sent. Three typical uses for this feature can be distinguished. In the first case, the author(s) may not have regular machine-based mailboxes and therefore wish(es) to indicate an alternate machine address. In the second case, an author may wish additional persons to be made aware of, or responsible for, responses; responders should send their replies to the "Reply-To" mailbox(es) listed in the original message. A somewhat different use may be of some help to "text message teleconferencing" groups equipped with automatic distribution services: include the address of that service in the "Reply-To" field of all messages submitted to the teleconference; then participants can "reply" to conference submissions to guarantee the correct distribution of any submission of their own.

Reply-To fields are NOT required to contain any machine addresses (i.e., host-phrases). Note, however, that the absence of even one valid network address will tend to prevent software systems from automatically assisting users in conveniently responding to mail.

NOTE: For systems which automatically generate address lists for replies to messages, the following recommendations are made:

- o The receiver, when replying to a message, should NEVER automatically include the "Sender" host-phrase in the reply's address list;
- o If the "Reply-To" field exists, then the reply should go ONLY to the addresses indicated in that field and not to the addresses indicated in the "From" field.

IV. Semantics

A. Address Fields

(Extensive examples are provided in Section V.) This recommendation is intended only for originator-fields and is not intended to suggest that replies should not also be sent to the other recipients of this message. It is up to the respective mail handling programs to decide what additional facilities will be provided.

3. Receiver Fields

a. To

This field contains the identity of the primary recipients of the message.

b. cc

This field contains the identity of the secondary recipients of the message.

b. Bcc

This field contains the identity of additional recipients of the message. The contents of this field are not included in copies of the message sent to the primary and secondary recipients. Some systems may choose to include the text of the "Bcc" field only in the author(s)'s copy, while others may also include it in the text sent to all those indicated in the "Bcc" list.

B. REFERENCE SPECIFICATION FIELDS

1. Message-ID

This field contains a unique identifier (the phrase) which refers to THIS version of THIS message. The uniqueness of the message identifier is guaranteed by the host which generates it. This identifier is intended to be machine readable and not necessarily meaningful to humans. A message identifier pertains to exactly one instantiation of a particular message; subsequent revisions to the message should each receive a new message identifier.

2. In-Reply-To

The contents of this field identify previous correspondence which this message answers. Note that if message identifiers are used in this field, they must use the mach-id specification format.

(476)

IV. Semantics

B. Reference Specification Fields

3. References

The contents of this field identify other correspondence which this message references. Note that if message identifiers are used, they must use the mach-id specification format.

4. Keywords

This field contains keywords or phrases, separated by commas.

C. OTHER FIELDS AND SYNTACTIC ITEMS

1. Subject

The "Subject" field is intended to provide as much information as necessary to adequately summarize or indicate the nature of the message.

2. Comments

Permits adding text comments onto the message without disturbing the contents of the message's body.

3. Extension-field

A relatively limited number of common fields have been defined in this document. As network mail requirements dictate, additional fields may be standardized. The authors of this document will regulate the publishing of such definitions as extensions to the basic specification.

4. User-defined-field

Individual users of network mail are free to define and use additional header fields. Such fields must have names which are not already used in the current specification or in any definitions of extension-fields, and the overall syntax of these user-defined-fields must conform to this specification's rules for delimiting and folding fields. Due to the extension-field publishing process, the name of a user-defined-field may be pre-empted.

(477)

IV. Semantics

D. Dates

D. DATES AND TIMES

If included, day-of-week must be the day implied by the date specification.

Time zone may be indicated in several ways. The military standard uses a single character for each zone. "Z" is Greenwich Mean Time; "A" indicates one hour earlier, and "M" indicates 12 hours earlier; "N" is one hour later, and "Y" is 12 hours later. The letter "J" is not used. The other remaining two forms are taken from ANSI standard X3.51-1975. One allows explicit indication of the amount of offset from GMT; the other uses common 3-character strings for indicating time zones in North America.

V. Examples

A. Addresses

V. EXAMPLES

A. ADDRESSES

1. Alfred E. Neuman <Neuman at BBN-TENEXA>

2. Neuman@BBN-TENEXA

These two "Alfred E. Neuman" examples have identical semantics, as far as the operation of the local host's mail sending (distribution) program (also sometimes called its "mailer") and the remote host's FTP server are concerned. In the first example, the "Alfred E. Neuman" is ignored by the mailer, as "Neuman at BBN-TENEXA" completely specifies the recipient. The second example contains no superfluous information, and, again, "Neuman@BBN-TENEXA" is the intended recipient.

3. Al Neuman at BBN-TENEXA

This is identical to "Al Neuman <Al Neuman at BBN-TENEXA>". That is, the full phrase, "Al Neuman", is passed to the FTP server. Note that not all FTP servers accept multi-word identifiers; and some that do accept them will treat each word as a different addressee (in this case, attempting to send a copy of the message to "Al" and a copy to "Neuman").

4. "George Lovell, Ted Hackle" <Shared-Mailbox at Office-1>

This form might be used to indicate that a single mailbox is shared by several users. The quoted string is ignored by the originating host's mailer, as "Shared-Mailbox at Office-1" completely specifies the destination mailbox.

4. Wilt (the Stilt) Chamberlain at NBA

The "(the Stilt)" is a comment, which is NOT included in the destination mailbox address handed to the originating system's mailer. The address is the string "Wilt Chamberlain", with exactly one space between the first and second words. (The quotation marks are not included.)

V. Examples

B. Address Lists

B. ADDRESS LISTS

Gourmets: Pompous Person <WhoZiWhatZit at Cordon-Bleu>,
Cooks: Childs at WGBH, Galloping Gourmet at
ANT (Australian National Television);,
Wine Lovers: Cheapie at Discount-Liquors,
Port at Portugal;;.

Jones at SEA

This group list example points out the use of comments, the nesting of groups, and the mixing of addresses and groups. Note that the two consecutive semi-colons preceding "Jones at SEA" mean that Jones is NOT a member of the Gourmets group.

C. ORIGINATOR ITEMS

1. Author-sent

George Jones logs into his Host as "Jones". He sends mail himself.

From: Jones at Host
or
From: George Jones <Jones at Host>

2. Secretary-sent

George Jones logs in as Jones on his Host. His secretary, who logs in as Secy on Shost sends mail for him. Replies to the mail should go to George, of course.

From: George Jones <Jones at Host>
Sender: Secy at SHost

3. Shared directory or unrepresentative directory-name

George Jones logs in as Group at Host. He sends mail himself; replies should go to the Group mailbox.

From: George Jones <Group at Host>

V. Examples

C. Originator Items

4. Secretary-sent, for user of shared directory

George Jones' secretary sends mail for George in his capacity as a member of Group while logged in as Secy at Host. Replies should go to Group.

From: George Jones<Group at Host>
Sender: Secy at Host

Note that there need not be a space between "Jones" and the "<", but adding a space enhances readability (as is the case in other examples).

5. Secretary acting as full agent of author

George Jones asks his secretary (Secy at Host) to send a message for him in his capacity as Group. He wants his secretary to handle all replies.

From: George Jones <Group at Host>
Sender: Secy at Host
Reply-To: Secy at Host

6. Agent for user without online mailbox

A non-ARPANET user friend of George's, Sarah, is visting. George's secretary sends some mail to a friend of Sarah in computer-land. Replies should go to George, whose mailbox is Jones at Host.

From: Sarah Friendly
Sender: Secy at Host
Reply-To: Jones at Host

7. Sent by member of a committee

George is a member of a committee. He wishes to have any replies to his message go to all committee members.

From: George Jones
Sender: Jones at Host
Reply-To: Big-committee: Jones at Host,
Smith at Other-Host,
Doe at Somewhere-Else:

Note that if George had not included himself in the enumeration of Big-committee, he would not have gotten an implicit reply; the presence of the "Reply-co" field SUPERSEDES the sending of a reply to the person named in the "From" field.

V. Examples

C. Originator Items

8. Example of INCORRECT use

George desires a reply to go to his secretary; therefore his secretary leaves his mailbox address off the "From" field, leaving only his name, which is not, itself, a mailbox address.

From: George Jones
Sender: Secy at SHost

THIS IS NOT PERMITTED. Replies are NEVER implicitly sent to the "Sender"; George's secretary should have used the "Reply-To" field, or the mail creating program should have forced the secretary to.

9. Agent for member of a committee

George's secretary sends out a message which was authored jointly by all the members of the "Big-committee".

From: Big-committee: Jones at Host,
Smith at Other-Host,
Doe at Somewhere-Else;
Sender: Secy at SHost

D. COMPLETE HEADERS

1. Minimum required:

Date: 26 August 1976 1429-EDT
From: Jones at Host

2. Using some of the additional fields:

Date: 26 August 1976 1430-EDT
From: George Jones<Group at Host>
Sender: Secy at SHOST
To: Al Neuman at Mail-Host,
Sam Irwin at Other-Host
Message-ID: <some string at SHOST>

V. Examples

D. Complete Headers

3. About as complex as you're going to get:

Date : 27 Aug 1976 0932-PDT
From : Ken Davis <KDavis at Other-Host>
Subject : Re: The Syntax in the RFC
Sender : KSecy at Other-Host
Reply-To : Sam Irving at Other-Host
To : George Jones <Group at Host>,
Al Neuman at Mad-Host
cc : Important folk:
Tom Softwood <Balsa at Another-Host>,
Sam Irving at Other-Host;,
Standard Distribution::Include:
 </main/davis/people/standard at Other-Host,
 "<Jones>standard.dist.3" at Tops-20-Host>.
 (The following Included Postal list is part
 of Standard Distribution.)
 :Postal::Include: Non-net-addr@Other-host;,
 :Postal: "Sam Irving, P.O. Box 001, Las Vegas,
 Nevada" (So that he can stay
 apprised of the situation)
Comment : Sam is away on business. He asked me to handle
his mail for him. He'll be able to provide a
more accurate explanation when he returns
next week.
In-Reply-To: <some string at SHOST>
Special (action): This is a sample of multi-word field-
names, using a range of characters. There
could also be a field-name "Special (info)".
Message-ID: <4231.629.XYzi-What at Other-Host>

APPENDIX

A. ALPHABETICAL LISTING OF SYNTAX RULES

pre>
address = host-phrase / quoted-string
 / (*phrase "<" #address ">")
 / (*phrase ":" #address ";")
 / (":" ("Include" / "Postal" / atom) ":" address)
ALPHA = <any TELNET ASCII alphabetic character>
atom = 1*<any CHAR except specials and CTLs>

CHAR = <any TELNET ASCII character>
comment = "(" *(ctext / comment / quoted-pair) ")"
CR = <TELNET ASCII carriage return>
CRLF = CR LF
ctext = <any CHAR excluding "(", ")", CR, LF and
 including linear-white-space>
CTL = <any TELNET ASCII control character and DEL>

date = 1*2DIGIT ["-"] month ["-"] (2DIGIT /4DIGIT)
date-field = "Date" ":" date-time
date-time = [day-of-week ", "] date time
day-of-week = "Monday" / "Mon" / "Tuesday" / "Tue"
 / "Wednesday" / "Wed" / "Thursday" / "Thu"
 / "Friday" / "Fri" / "Saturday" / "Sat"
 / "Sunday" / "Sun"

delimiters = specials / comment / linear-white-space
DIGIT = <any TELNET ASCII digit>

extension-field = <Any field which is defined in a document
 published as a formal extension to this
 specification>

field = field-name ":" [field-body] CRLF

fields = date-field originator-fields *optional-field
field-body = field-body-contents
 [CRLF LWSP-char field-body]
field-body-contents = <the TELNET ASCII characters making up the
 field-body, as defined in the following sections,
 and consisting of combinations of atom, quoted-
 string, and specials tokens, or else consisting of
 texts>
field-name = fntom *(LWSP-char [fntom])
fntom = 1*<any CHAR, excluding CTLs, SPACE, and ":">

```

host-indicator = 1*( "at" / "@" ) node )
host-phrase = phrase host-indicator
hour         = 2DIGIT [ ":" ] 2DIGIT [ [ ":" ] 2DIGIT ]
HTAB        = <TELNET ASCII horizontal-tab>

LF          = <TELNET ASCII linefeed>
linear-white-space = 1*([CRLF] LWSP-char)
LWSP-char   = SPACE / HTAB

mach-id      = "<" host-phrase ">"
mailbox      = host-phrase / (phrase mach-id)
message      = fields *(CRLF *text)
month        = "January" / "Jan" / "February" / "Feb"
              / "March" / "Mar" / "April" / "Apr"
              / "May" / "June" / "Jun"
              / "July" / "Jul" / "August" / "Aug"
              / "September" / "Sep" / "October" / "Oct"
              / "November" / "Nov" / "December" / "Dec"

node         = word / 1DIGIT

optional-field =
    / "To" ":" #address
    / "cc" ":" #address
    / "bcc" ":" #address
    / "Subject" ":" *text
    / "Comments" ":" *text
    / "Message-ID" ":" mach-id
    / "In-Reply-To" ":" #(phrase / mach-id)
    / "References" ":" #(phrase / mach-id)
    / "Keywords" ":" #phrase
    / extension-field
    / user-defined-field
originator-fields =
    ( "From" ":" mailbox
      [ "Reply-To" ":" #address ] )
    / ( "From" ":" 1#address
        "Sender" ":" mailbox
        [ "Reply-To" ":" #address ] )

phrase       = 1*word

quoted-pair  = "\" CHAR
quoted-string = "<" *(qtext / quoted-pair) ">"
qtext        = <any CHAR except "<", CR, or LF and including
              linear-white-space>
SPACE        = <TELNET ASCII space>
specials     = "(" / ")" / "<" / ">" / "@" / "," / ";" / ":"
              / "\" / "<"

text         = <any CHAR, including bare CR and/or bare LF, but
              NOT including CRLF>

```

Standard for the Format of Text Messages
Appendix
A. Alphabetical Listing of Syntax Rules

33

time = hour zone

user-defined-field = <Any field which has not been defined in
 this specification or published as an extension to
 this specification; names for such fields must be
 unique and may be preempted by published
 extensions>

word = atom / quoted-string

zone = (("+" / "-") 4DIGIT)
 / (["-"] (1ALPHA
 / "GMT" / "NST" / "AST" / "ADT" / "EST" / "EDT"
 / "CST" / "CDT" / "MST" / "MDT" / "PST" / "PDT"
 / "YST" / "YDT" / "HST" / "HDT" / "BST" / "BDT"))

<"> = <TELNET ASCII quote mark>

B. SIMPLE PARSING

Some mail-reading software systems may wish to perform only minimal processing, ignoring the internal syntax of structured field-bodies and treating them the same as unstructured-field-bodies. Such software will need only to distinguish:

- Header fields from the message body.
- Beginnings of fields from lines which continue fields.
- Field-names from field-contents.

The abbreviated set of syntactic rules which follows will suffice for this purpose. They describe a limited view of messages and are a subset of the syntactic rules provided in the main part of this specification. One small exception is that the contents of field-bodies consist only of text:

SYNTAX:

```
message      = *field *(CRLF *text)
field        = field-name ":" [field-body] CRLF
field-name   = fnatom *( LWSP-char [fnatom] )
fnatom       = 1*<any CHAR, excluding CTLs, SPACE, and ":">
field-body   = *text [CRLF LWSP-char field-body]
```

SEMANTICS:

Headers occur before the message body and are terminated by a null line (i.e., two contiguous CRLFs).

A line which continues a header field begins with a SPACE or HTAB character, while a line beginning a field starts with a printable character which is not a colon.

A field-name consists of one or more printable characters (excluding colon), each separated by one or more SPACES or HTABS. A field-name MUST be contained on one line. Upper and lower case are not distinguished when comparing field-names.

BIBLIOGRAPHY

- ANSI. Representations of universal time, local time differentials, and United States time zone references for information interchange. ANSI X3.51-1975; American National Standards Institute: New York, 1975.
- Bhushan, A.K. The File Transfer Protocol. ARPANET Request for Comments, No. 354, Network Information Center No. 10596; Augmentation Research Center, Stanford Research Institute: Menlo Park, July 1972.
- Bhushan, A.K. Comments on the File Transfer Protocol. ARPANET Request for Comments, No. 385, Network Information Center No. 11357; Augmentation Research Center, Stanford Research Institute: Menlo Park, August 1972.
- Bhushan, A.K., Pogram, K.T., Tomlinson, R.S., and White, J.E. Standardizing Network Mail Headers. ARPANET Request for Comments, No. 561, Network Information Center No. 18516; Augmentation Research Center, Stanford Research Institute: Menlo Park, September 1973.
- Feinler, E.J. and Postel, J.B. ARPANET Protocol Handbook. Network Information Center No. 7104; Augmentation Research Center, Stanford Research Institute: Menlo Park, April 1976. (NTIS AD A003890).
- McKenzie, A. File Transfer Protocol. ARPANET Request for Comments, No. 454, Network Information Center No. 14333; Augmentation Research Center, Stanford Research Institute: Menlo Park, February 1973.
- McKenzie, A. TELNET Protocol Specification. Network Information Center No. 18639; Augmentation Research Center, Stanford Research Institute: Menlo Park, August 1973.
- Myer, T.H. and Henderson, D.A. Message Transmission Protocol. ARPANET Request for Comments, No. 680, Network Information Center No. 32116; Augmentation Research Center, Stanford Research Institute: Menlo Park, 1975.
- Neigus, N. File Transfer Protocol. ARPANET Request for Comments, No. 542, Network Information Center No. 17759; Augmentation Research Center, Stanford Research Institute: Menlo Park, July 1973.
- Pogram, K., Vittal, J., Crocker, D. and Henderson, A. Proposed official standard for the format of ARPA network messages.

ARPANET Request for Comments, No. 724, Network Information Center No. 37435; Augmentation Research Center, Stanford Research Institute: Menlo Park, May 1977.

Postel, J.B. Revised FTP Reply Codes. ARPANET Request for Comments, No. 640, Network Information Center No. 30843; Augmentation Research Center, Stanford Research Institute: Menlo Park, June 1974.

C/30 INFORMATION

March 1982

(493)

C/30 PACKET SWITCH

I. INTRODUCTION

This memo describes the performance, function, and cost, of the BBN C/30 packet switches being used to reconfigure the ARPANET backbone. Whenever possible, the information furnished in this memo is based on actual operational experience with the ARPANET technology.

II. SWITCH PARAMETERS

A. Cost

The C/30 packet switch has a modular physical architecture that permits variations in the hardware as well as the software configuration. In particular, the I/O configuration can be tailored using a mixture of I/O boards and even partial population of individual I/O boards.

The basic C/30 packet switch less I/O hardware consists of the following components:

QTY.	PART#	DESCRIPTION	COST
1	5308	Expanded C/30 processor with 32Kw	\$18,000
1	5352	2nd 32Kw memory expansion	2,100

			\$20,100
		less 8% government disc	-1,608

			\$18,492
1		30 CPS hardcopy control terminal	1,248

			\$19,740

The C/30 packet switch can be configured with any mix (up to five 5 boards) of the following two types of I/O boards:

PART #	DESCRIPTION	COST (-8%)
5401	I/O board with 4 1822 ports and 6 bisync ports	\$4,140
5405	I/O board with 16 HDLC ports	\$5,428

The 5401 I/O board is the original IMP I/O board and is used to provide 1822 host access ports and H316/516-compatible bisync trunk ports. The 5405 I/O board provides the capability of interfacing bit-stuffed (HDLC framing and transparency) hosts and trunks.

Each I/O board must also be populated with line adapters consonant with the actual number and type of hosts and trunks to be attached. The adapters that are available for the 5401 board service individual ports, while the ones for the 5405 service a group of eight ports. The line adapters relevant to this memo are:

PART #	DESCRIPTION	COST (-8%)
-----	-----	-----
5442	Adapter for single 1822 host on 5401 board	\$276
5443	Adapter for single MIL 188-114 bisync line on 5401	\$230
5424	Adapter for eight MIL 188-114 bit-stuffed lines on 5405	\$736

The C/30 packet switch is designed to be mounted in a standard 19-in. rack, with a maximum of 2 C/30s per rack. The cost of a rack is \$1288.

The above information is sufficient to derive the cost of any C/30 configuration. For illustrative purposes, we have produced three sample configurations.

CONFIGURATION A: IMP with 4 HDH (HDLC) hosts and 3 bit-stuffed synchronous trunks:

QTY.	PART	COST (-8%)
----	----	-----
1	Basic IMP	\$19,740
1	5405 I/O board	5,428
1	542X line adapter (8 lines) for HDH and trunk	736
1	Rack	1,288

		\$27,192

CONFIGURATION B: IMP with 2 1822 hosts and 3 bisync trunks:

QTY.	PART	COST (-8%)
----	----	-----
1	Basic IMP	\$19,740
1	5401 I/O board	4,140
2	5442 1822 adapter for host	552
3	5433 line adapter (1 line) for trunk	690
1	Rack	1,288

		\$26,410

CONFIGURATION C: IMP with 6 1822 hosts, 20 HDH hosts, 3 VDH hosts, and 8 bit-stuffed synchronous trunks.

QTY.	PART	COST (-8%)
----	----	-----
1	Basic IMP	\$19,740
2	5401 I/O board	8,280
2	5405 I/O board	10,856
6	5442 1822 adapter for host	1,656
4	542X line adapter (8 lines) for HDH and trunk	2,944
3	5433 line adapter (1 line) for VDH	690
1	Rack	1,288

		\$45,454

There is also a TEMPEST/HEMP option for the C/30 packet switch. This option is being developed under contract to DCA. We have estimated the additional cost per C/30 in production quantities to be about \$20,000.

All prices are based upon the current Requirements Contract with DECCO.

B. Performance

1. Throughput

A C/30 has been measured processing 300 packets/second (for tandem traffic, where 1 packet processing includes input on one trunk and output on another). The measurement also determined that the limitation was not CPU capacity, but interrupt response time for the emulated H316 I/O structure. We are implementing a new buffered I/O structure in firmware which will eliminate virtually all I/O latency constraints from the software. We have estimated the capacity of the C/30 under these improved conditions to be in the neighborhood of 500 packets/second.

We have also measured originating and arriving throughput capacity for host traffic, and it was subject to the limitations of the trunk capacity, which are in turn governed by the latency constraints described above. Individual host traffic streams have been measured in excess of 100 packets/second full duplex (100 packets in each direction simultaneously.)

2. Delay

An integral aspect of the ARPANET routing algorithm is a continuous measurement of the delay experienced by packets as they traverse each packet switch. The algorithm computes the delay from when a packet is first received at a switch to when it is received at each one of its neighbors via the interswitch trunks. This delay has four components: packet processing, queueing, transmission, and propagation. The propagation delay is purely a function of the speed-of-light distance to the next switch. The transmission delay is purely a function of the speed of the trunk and the size of the packet. The queueing delay is purely a function of the number and size of the preceding packets and is governed by traffic loading. The packet processing delay is the only component that is a function of the packet switch capacity, and can be computed by subtracting off the other three components from measured values.

In practice, only the propagation delay can be directly computed. Transmission delay must be computed using average packet sizes. Queueing and processing delay cannot be easily separated, except by examining paths that are known to carry little traffic and hence presumably experience no queueing delay. At a lower bound, we have measured pure processing delay of less than 1 millisecond, at nodes with known low traffic flows. For a typical measurement from the DEC to the LINCOLN, ARPANET was 6.4 milliseconds, of which .8 ms are the propagation delay used by the routing algorithm. This includes transmission, queueing, and processing. The speed is 50 kb, and even a 0 data-bit packet consumes 3.2 transmission delay, (a 200 data-bit packet consumes 7.2 ms, and a maximum 1008 data-bit packet 23.8 ms.). As another example, the average delay from NPS to SCOTT, probably the most highly loaded path in the ARPANET, was 32 ms., of which 14.4 ms. were purely propagation delay, leaving 17.6 ms. for the sum of transmission, queueing, and processing.

Since the processing of tandem traffic is carried out mostly by processes at the highest priority level, the processing delay is only a weak function of CPU loading due to other (host) traffic. A conservative figure for processing delay is 1-2 ms. for a light to moderately loaded switch and 2-3 ms. for a heavily loaded one.

C. Configuration

The maximum configuration of a C/30 packet switch is governed by the number and type of I/O boards as described in II-A. The C/30 chassis will accommodate a maximum of five I/O boards. If these boards are all HDLC (5405 type), this yields a maximum of $16 \times 5 = 80$ ports for use as host or trunk lines. On the other hand, if the boards are all 5401 type, that yields a maximum of $4 \times 5 = 20$ 1822 host ports and $6 \times 5 = 30$ VDH or trunk ports. Of course combinations of the two board types are also possible.

D. Transmission Constraints

1. Transmission Speed

The C/30 packet switch currently supports synchronous full-duplex lines from 4.8 to 230 Kb. Operation at line speeds in excess of 100 Kb is currently marginal at higher loads due to the latency constraints inherited from the H316 architecture. The new firmware buffered I/O structure will eliminate this constraint and will permit the C/30 to operate the > 100 Kb lines subject only to overall packet processing capacity.

2. Propagation Delay

High-propagation delay lines require an acknowledgement window sufficiently large to accommodate a full round-trip's time worth of packets. The link protocol permits up to 128 packets in flight simultaneously, and follows the ARPANET technique of using independent acknowledgement channels so that only missed packets are retransmitted.

It should be noted that while the link protocols are designed to take full advantage of the bandwidth of high-propagation delay lines, the routing algorithm will nonetheless continue to select the minimum delay route to every destination. Thus, such lines (e.g. satellite links) can only be practically used when alternative routes also use similar links (e.g. connecting CONUS to Europe or Hawaii).

E. Other Considerations

1. Manning

The C/30 packet switch is designed for unattended operation. All control, software maintenance and most fault isolation and diagnosis is done remotely via down-line access from one or more NOCs (Network Operation Centers). Over ten years of experience with the ARPANET, where packet switches have been located in extremely diverse environments, has produced operational techniques which require very minimal site assistance. In the few cases, outside of hardware malfunction and repair, when there is need for site assistance, we are able to utilize untrained personnel (including in some cases janitors and security guards) to bring the node back under control of the NOC.

2. Reliability and Availability

The C/30 was designed, utilizing our past experience with operating packet switches, to be a robust piece of hardware. The thermal design is extremely conservative, with low board densities and considerable air space between boards. There are no moving parts (except for the bootload cassette). There is no backplane, only flexible cabling between boards, and connections are via standard pin-socket connectors. In short, it is a modern, understressed, low-maintenance design which is intended to provide high MTBF. Since the C/30 consists of up to seven boards (processor, power, memory, and I/O) and their connecting cables, hardware fault isolation is performed by board swap rather than analysis of individual electronic components, thus minimizing MTTR.

We have a developing picture of the ultimate reliability and availability of C/30s based on actual ARPANET installations. From March through September 1981, 19 C/30 IMPs have been installed on the ARPANET, and their average availability in the most recent month has already exceeded the average ARPANET node (despite the normal teething problems associated with new installations and early production of new equipment). For the month of September, the average availability (excluding one node installed that month) was 99.93%, with a total of 20 outages due to hardware/software failures for 18 C/30s, yielding an average MTBF of 640 hours and an MTRR of 28 minutes. There is ample evidence to expect that the long-term availability of C/30s will consistently exceed 99.9%.

3. Future Growth

The C/30 is a member of the MBB (Microprogrammable Building Block) family. As such, it could be field-upgraded to versions of the MBB that directly support the C higher-level language and several megabytes of memory. This is an important advantage should the IMP packet switching program someday be rewritten in the higher level language for the sake of better maintainability, more modularity, and added functionality.

III. NETWORK PARAMETERS

A. Routing and Control Strategy

The C/30 packet switch software uses the latest ARPANET routing algorithm. [See "The New Routing Algorithm for the ARPANET," McQuillan et al., IEEE Transactions on Communications, May 1980.] This algorithm is a dynamic, adaptive procedure which determines the least delay path to all destinations and routes traffic accordingly. Line and CPU overhead are less than two percent, most nodes react to changes in topology or significant changes in delay within 100 ms, and the algorithm detects local congestion and routes packets around congested areas.

The routing algorithm is implemented as a distributed process within the switching nodes and does not rely on the NCC or other manual intervention to operate. Nodes and trunks may be added or removed from the network and the routing algorithm will automatically adapt to the new topology, eliminating a potentially large class of problems in network configuration control. Since the algorithm is purely delay-oriented, and since the individual link delays are computed based on actual measured packet delays, the network topology can accommodate variations in circuit types, including low or high bandwidth and low or high propagation delay, as long as the configuration is consistent with a minimum-delay routing strategy.

In a network with homogeneous trunks (e.g. all 50 kb land lines), the routing algorithm can be modelled approximately by a simple min-hop routing scheme.

B. Security Considerations

The C/30 packet switch is designed to operate at a single security level. The software does not permit spoofing by the subscribers and contains some provisions to minimize denial of service problems. A military secure C/30 network would run at TS level, have physical security for each packet switch, and bulk-encrypt the interswitch trunks. There would also be link encryption of subscriber (host) access lines as appropriate. Users at security levels higher or lower than TS would communicate over the TS net using end-to-end encryption devices (IPLIs) which would provide segregated communities of intercommunicating users at the same security level and sharing the same encryption key. In addition, the node software provides a facility for assigning each subscriber to one or more compartmentalization groups and disallows communication between subscribers who do not have at least one such group in common.

C. Precedence and Preemption

The C/30 software currently provides two levels of precedence and no preemption capability. We have studied the requirements for precedence and preemption for a military net and concluded that with a small amount of modification the C/30s could meet or exceed virtually all the requirements. It would be fairly straight-forward to add more levels of precedence, since the basic mechanism for differentiating and servicing multiple levels already exists. We can also meet the requirements for non-blocking of high-priority traffic by pre-allocating the necessary resources (which are reasonably abundant in a 64k C/30 IMP) for such traffic, thus eliminating the need to preempt the resources used by other traffic.

E. Interfacing

The C/30 packet switch offers a variety of physical and logical host/switch interfacing arrangements.

At the physical link level, there is a choice of asynchronous four-way handshake (referred to as 1822 hardware, documented in BBN Report 1822), byte-stuffing asynchronous (biaync) with 24-bit CRC, and bit-stuffing synchronous (HDLC) with 16-bit CRC. The 1822 link is assumed to be error free and is limited to 2,000 feet. The synchronous links can be any communications line of unlimited length. The synchronous links support speeds from 4.8 to 230 Kb.

The C/30 currently supports the standard ARPANET host/IMP interface, either directly through 1822 hardware or via synchronous link using the HDH (HDLC) or VDH (bysync) protocols. The ARPANET host/IMP protocol supports both virtual circuit and pure datagram service, and is being enhanced to provide a general logical addressing facility (for multi-homing and multi-naming of hosts.)

Because the C/30 features microprogrammed I/O, it is also feasible to develop additional interfacing options (e.g. ADCCP with 32-bit CRC) if required, without additional hardware development.

Elizabeth Feinler
Ken Harrenstien
Zaw-Sing Su
Vic White
Network Information Center
SRI International

RFC 810
1 March 1982
References: RFC 811, 796
Obsoletes: RFC 608

DoD INTERNET HOST TABLE SPECIFICATION

INTRODUCTION

The ARPANET Official Network Host Table, as outlined in RFC 608, no longer suits the needs of the DoD community, nor does it follow a format suitable for internetting. This paper specifies a new host table format applicable to both ARPANET and Internet needs.

In addition to host name to host address translation and selected protocol information, we have also included network and gateway name to address correspondence, and host operating system information.

This Host Table is utilized by the DoD Host Name Server maintained by the ARPANET Network Information Center (NIC) on behalf of the Defense Communications Agency (DCA) (RFC 811). It obsoletes the host table described in RFC 608.

LOCATION OF THE STANDARD DoD ONLINE HOST TABLE

A machine-translatable ASCII text version of the new DoD Host Table is online in the file <NETINFO>HOSTS.TXT on the SRI-NIC host. It can be obtained by connecting to host SRI-NIC (10.0.0.73) from your local FTP server, logging in as user=ANONYMOUS, password=GUEST, and doing a 'get' on <NETINFO>HOSTS.TXT. The same table may also be obtained via the NIC Host Name Server.

NOTE: See Appendix A. for timeframe for cutover.

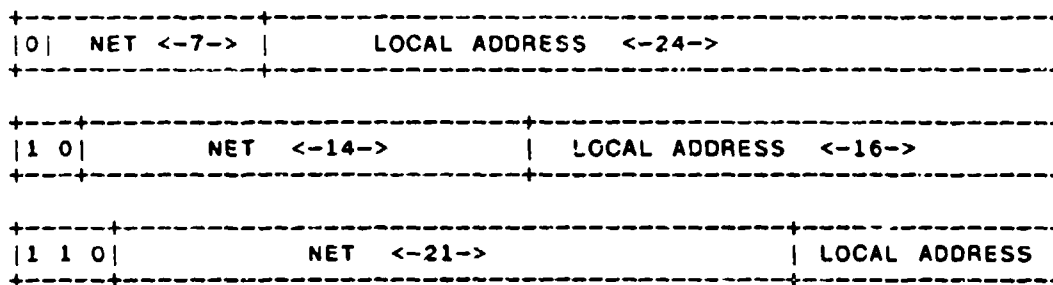
ASSUMPTIONS

1. A "name" (Net, Host, Gateway, or Domain name) is a text string up to 24 characters drawn from the alphabet (A-Z), digits (0-9), and the minus sign (-) and period (.). No blank or space characters are permitted as part of a name. No distinction is made between upper and lower case. The first character must be a letter. The last character must not be a minus sign or period. A host which serves as a GATEWAY should have "-GATEWAY" or "-GW" as part of its name. A host which is a TIP or a TAC should have "-TIP" or "-TAC" as part of its host name, if it is an ARPANET or DoD host.

2. Internet Addresses are 32-bit addresses (RFC 796). In the host table described herein each address is represented by four decimal numbers separated by a period. Each decimal number represents 1 octet.

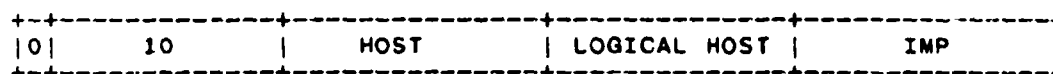
3. If the first bit of the first octet of the address is 0 (zero), then the next 7 bits of the first octet indicate the network number (Class A Address). If the first two bits are 1,0 (one,zero), then the next 14 bits define the net number (Class B Address). If the first 3 bits are 1,1,0 (one,one,zero), then the next 21 bits define the net number (Class C Address) (RFC 796).

This is depicted in the following diagram:



4. The LOCAL ADDRESS portion of the internet address identifies a host within the network specified by the NET portion of the address.

5. For the ARPANET (a Class A network), the NET address is 10 (decimal) and the LOCAL ADDRESS maps as follows: the second octet defines the physical host, the third octet defines the logical host, and the fourth defines the IMP.



(NOTE: RFC 796 describes the local address mappings for several other networks.)

6. It is the responsibility of the user using this host table to translate it into whatever format is needed for his or her purposes.

7. Names and Addresses for DoD networks, gateways, and hosts will be negotiated and registered with the Network Information Center (NIC@SRI-NIC or (415) 859-4775) before being used and before traffic is passed by a DoD host. For an interim period the NIC will attempt to keep similar information for non-DoD networks and hosts if this

information is provided, and as long as it is needed, i.e., until intercommunicating network name servers are in place.

EXAMPLE OF NEW HOST TABLE FORMAT

```
NET : 10.0.0.0 : ARPANET :  
NET : 18.0.0.0 : LCSNET :  
GATEWAY : 10.0.0.77, 18.8.0.4 : MIT-GW : MOS : IP/GW :  
HOST : 10.0.0.73 : SRI-NIC,NIC : TENEX : NCP/TELNET, NCP/FTP,  
          TCP/TELNET, TCP/FTP :  
HOST: 10.2.0.11 : SU-TIP,FELT-TIP :::
```

SYNTAX AND CONVENTIONS

;	(semicolon)	is used to denote the beginning of a comment. Any text on a given line following a ';' is comment, and not part of the host table.
NET		keyword introducing a network name/address entry
GATEWAY		keyword introducing a gateway name/address entry
HOST		keyword introducing a host name/address entry
:	(colon)	is used as a field delimiter
::	(2 colons)	indicates a null field
,	(comma)	is used as a data element delimiter
XXX/YYY		indicates protocol information of the type TRANSPORT/SERVICE.

where TRANSPORT/SERVICE options are specified as

"FOO/BAR"	- both transport and service known
"FOO"	- transport known; services not known or not running. OR
"BAR"	- name is known, what it does is not

NOTE: See Appendices B and C for specific options and acronyms.

Each host table entry is an ASCII text string comprised of 6 fields, where

Field 1	= KEYWORD indicating whether this entry pertains to a NET, GATEWAY, or HOST. NET entries cannot have alternate addresses or nicknames.
Field 2	= Internet Address of Network, Gateway, or Host

followed by alternate addresses
 Field 3 = Official Name of Network, Gateway, or Host
 (with optional nicknames)
 Field 4 = Operating System
 Field 5 = Protocol List

Fields 4 and 5 are optional.

Fields 3-5, if available, pertain to the first address in Field 2.

'Blanks' (spaces and tabs) are ignored between data elements or fields, but are disallowed within a data element.

Each entry ends with a colon.

The host table will be sorted by internet address.

GRAMMATICAL HOST TABLE SPECIFICATION

A. Parsing grammar

```
<entry> ::= <keyword> ":" <addresses> ":" <names> [ ":" [ <opsys> ]
           [ ":" [ <protocol list> ] ] ] ":"
<addresses> ::= <address> * [ "," <address> ]
<address> ::= <octet> "." <octet> "." <octet> "." <octet>
<octet> ::= <0 to 255 decimal>
<names> ::= <netname> | <gatename>
           | <official hostname> * [ "," <nicknames> ]
<netname> ::= <name>
<gatename> ::= <name>
<official hostname> ::= <name>
<nickname> ::= <name>
<protocol list> ::= <protocol spec> * [ "," <protocol spec> ]
<protocol spec> ::= <transport name> "/" <service name> |
                   <raw protocol name>
```

B. Lexical grammar

```
<entry-field> ::= <entry-text> [ <cr> <lf> <blank> <entry-field> ]
<blank> ::= <space or tab>
<keyword> ::= NET | GATEWAY | HOST
<name> ::= <letter> [ * [ <letter-or-digit-or-hyphen> ] <letter-or-digit> ]
<opsys> ::= ITS | MULTICS | TOPS20 | UNIX...etc.
<transport name> ::= TCP | NCP | UDP | IP...etc.
<service name> ::= TELNET | FTP | SMTP | MTP...etc.
<raw protocol name> ::= <name>
<comment> ::= ";" <arbitrary text> <cr> <lf>
```

Notes:

1. Zero or more 'blanks' between separators " , : " are allowed. 'Blanks' are spaces and tabs.
2. Continuation lines are lines that begin with at least one blank. They may be used anywhere 'blanks' are legal to split an entry across lines.

BIBLIOGRAPHY

1. Feinler, E. and Kudlick, M. Host Names Online, RFC 608, Network Information Center, SRI International, Jan. 1973.
2. Postel, J. Assigned Numbers, RFC 790, Information Sciences Inst., Univ. of Southern Calif., Marina Del Rey, Sept. 1981.
3. Postel, J. Internet Protocol, RFC 791, Information Sciences Inst., Univ. of Southern Calif., Marina Del Rey, Sept. 1981.
4. Postel, J. Address Mappings, RFC 796, Information Sciences Inst., Univ. of Southern Calif., Marina Del Rey, Sept. 1981.
5. Feinler, E., Warrenstien, K., Su, Z. and White, V. Official DoD Internet Host Table Specification, RFC 810, Network Information Center, SRI International, March 1, 1982.

APPENDIX A. CUTOVER DETAILS

The cutover date for use of the new host table is 1 May 1982. The table below indicates which files will contain the old or the new versions of the host table for what period of time. After 1 August 1982, the old format for <NETINFO>HOSTS.TXT (specified in RFC-608) will no longer be supported.

May 1982	June-July 1982	August 1982 on
<NETINFO>HOSTS.TXT old version	<NETINFO>HOSTS.TXT new version	<NETINFO>HOSTS.TXT new version
<NETINFO>NHOSTS.TXT new version (test)	<NETINFO>NHOSTS.TXT new version	old version discontinued
<NETINFO>OHOSTS.TXT old version	<NETINFO>OHOSTS.TXT old version	

These periods of overlap should give implementors time to make the necessary changes to programs accessing this file.

APPENDIX B. TRANSPORT/SERVICE OPTIONS AND ACRONYMS

Current TRANSPORT/SERVICE options are:

IP	TCP/FTP
IP/GW	TCP/MTP
NCP	TCP/NNS
NCP/FTP	TCP/RJE
NCP/RJE	TCP/SMTP
NCP/SMTP	TCP/TELNET
NCP/TELNET	TCP/TFTP
NCP/NNS	UDP
NVP	
TCP	

Note: "TCP" implies IP is also implemented

Acronym definitions for the above protocol options are:

- FTP - File Transfer Protocol
- GW - Gateway Protocol
- IP - Internet Protocol
- MTP - Mail Transfer Protocol
- NCP - Network Control Protocol
- NNP - NIC Internet Name Server Protocol
- NVP - Network Voice Protocol
- RJE - Remote Job Entry Protocol
- SMTP - Simple Mail Transfer Protocol
- TELNET - TELNET Protocol
- TCP - Transmission Control Protocol
- TFTP - Trivial File Transfer Protocol
- UDP - User Datagram Protocol

APPENDIX C. OPERATING SYSTEM ACRONYMS

Current operating system acronyms are:

ASP	XRONOS	RSX11M	VMS
AUGUST	MCP	RT11	WAITS
BKY	MOS	SCOPE	
CCP	MPX-RT	SIGNAL	
DOS/360	MULTICS	SINTRAN	
ELF	MVT	TENEX	
EPOS	NOS	TOPS10	
EXEC-8	NOS/BE	TOPS20	
GCOS	OS/MVS	TSS	
GPOS	OS/MVT	UNIX	
ITS	RIG	VM/370	
INTERCOM	RSX11	VM/CMS	

Ken Harrenstien
Vic White
Elizabeth Feinler
Network Information Center
SRI International

RFC-811
1 March 1982

HOSTNAMES SERVER

INTRODUCTION

The NIC Internet Hostnames Server is an NCP/TCP-based host information program and protocol running on the SRI-NIC machine. It is one of a series of ARPANET/Internet name services maintained by the Network Information Center (NIC) at SRI International on behalf of the Defense Communications Agency (DCA). The function of this particular server is to deliver machine-readable name/address information describing networks, gateways, hosts, and eventually domains, within the internet environment. As currently implemented, the server provides the information outlined in the DoD Internet Host Table Specification (RFC 810).

QUERY/RESPONSE FORMAT

The name server accepts simple text query requests of the form

<command key> <argument(s)> [<options>]

where square brackets ("[]") indicate an optional field. The command key is a keyword indicating the nature of the request. The defined keys are explained below.

The response, on the other hand, is of the form

<response key> : <rest of response>

where <response key> is a keyword indicating the nature of the response, and the rest of the response is interpreted in the context of the key.

COMMAND/RESPONSE KEYS

The currently defined keywords are:

Command Keys:

HNAME	(find entry with given name)
HADDR	(find entry with given address)
ALL	(return entire host table)

[Page 1]

(513)

Response Keys:

ERR (entry not found, nature of error follows)
NET (entry found, rest of entry follows)
GATEWAY (entry found, rest of entry follows)
HOST (entry found, rest of entry follows)
BEGIN (followed by multiple entries)
END (done with BEGIN block of entries)

More keywords will be added as new needs are recognized. A more detailed description of the allowed requests/responses will follow.

PROTOCOL

To access this server from a program, connect to service host (SRI-NIC)

TCP: port 101 decimal
NCP: socket 101 decimal for ICP

send the information query, and await the response.

Note: Care should be taken to interpret the nature of the reply (e.g. single record or multiple record), so that no confusion about the state of the reply results. An "ALL" request will likely return several hundred or more records of all types (see RFC 810), whereas "HNAME" or "HADDR" will usually return one HOST record, or "BEGIN:", list of host records, "END:": if there is more than one match.

QUERY/RESPONSE EXAMPLES

1. HNAME Query - Given a name, find the entry or entries that match the name. For example:

HNAME SRI-NIC <CRLF> ;where <CRLF> is a carriage return/
linefeed, and 'SRI-NIC' is a host name

The likely response is:

HOST : 10.0.0.73 : SRI-NIC,NIC : TENEX : NCP :

A response may stretch across more than one line. Continuation lines always begin with at least one space. For example:

HOST : 10.0.0.73 : SRI-NIC,NIC :
TENEX : NCP :

2. HADDR Query - Given an internet address (as specified in RFC 796) find the entry or entries that match that address.
For example:

```
HNAME 10.0.0.73 <CRLF> ;where <CRLF> is a carriage return/
                        linefeed, and '10.0.0.73' is a host
                        address
```

The likely response is the same as for the HNAME request:

```
HOST : 10.0.0.73 : SRI-NIC,NIC : TENEX : NCP :
```

3. ALL Query - Deliver the entire internet host table in a machine-readable form. For example:

```
ALL <CRLF> ;where <CRLF> is a carriage return/linefeed
```

The likely response is the keyword 'BEGIN' followed by a colon ':', followed by the entire internet host table in the format specified in RFC 810, followed by 'END:'. For example:

```
BEGIN:
NET : 10.0.0.0 : ARPANET :
NET : 18.0.0.0 : LCSNET :
GATEWAY : 10.0.0.77, 18.8.0.4 : MIT-GW : MOS : IP/GW :
HOST : 10.0.0.73 : SRI-NIC,NIC : TENEX : NCP/TELNET,
      NCP/FTP, TCP :
HOST : 10.2.0.11 : SU-TIP, FELT-TIP ::
END:
```

ERROR HANDLING

1. ERR Reply - may occur on any query, and should be permitted in any access program using the name server. Errors are of the form

```
ERR : <code> : <string> :
```

as in

```
ERR : NAMNFD : Name not found :
```

The error code is a unique descriptor, limited to 8 characters in length for any given error. It may be used by the access program to identify the error and, in some cases, to handle it automatically. The string is an accompanying message for a given error for that case where the access program simply logs the error message. Current codes and their associated interpretations are

NAMNFD	--	Name not found; name not in table
ADRNFD	--	Address not found; address not in table
ILLCOM	--	Illegal command; command key not recognized
TMPSYS	--	Temporary system failure, try again later

REMARKS

The host name server described above runs over a single global internet host name/address data base. This data base is an extension of the old ARPANET Hosts.txt file, and is being maintained by the NIC to provide continuity during the transition and expansion to the internet environment. We view the central administration of a global host name data base, along with this simple name server, as an interim solution on the way to a decentralized, distributed name/address translation service. The NIC welcomes your comments and suggestions for such an expanded service. Send comments to NIC@SRI-NIC.

REFERENCES

1. Feinler, E., Harrenstien, K., Su, Z. and White, V. Official DoD Internet Host Table Specification, RFC 810, Network Information Center, SRI International, March 1, 1982.
2. Postel, J. Address Mappings, RFC 796, Information Sciences Inst., Univ. of Southern Calif., Marina Del Rey, Sept. 1981.
3. Pickens, J., Feinler, E., and Mathis, J. The NIC Name Server, A Datagram-based Information Utility, Network Information Center, SRI International, July 1979.

Ken Harrenstien
Vic White
Network Information Center
SRI International

RFC-812
1 March 1982

NICNAME/WHOIS

INTRODUCTION

The NICNAME/WHOIS Server is an NCP/TCP transaction based query/response server, running on the SRI-NIC machine, that provides netwide directory service to ARPANET users. It is one of a series of ARPANET/Internet name services maintained by the Network Information Center (NIC) at SRI International on behalf of the Defense Communications Agency (DCA). The server is accessible across the ARPANET from user programs running on local hosts, and it delivers the full name, U.S. mailing address, telephone number, and network mailbox for ARPANET users.

This server, together with the corresponding Identification Data Base provides online directory look-up equivalent to the ARPANET Directory. DCA strongly encourages network hosts to provide their users with access to this network service.

WHO SHOULD BE IN THE DATA BASE

DCA requests that each individual with a directory on an ARPANET host, who is capable of passing traffic across the ARPANET, be registered in the NIC Identification Data Base. To register, send full name, middle initial, U.S. mailing address (including mail stop and full explanation of abbreviations and acronyms), ZIP code, telephone (including Autovon and FTS, if available), and one network mailbox, via electronic mail to NIC@SRI-NIC.

PROTOCOL

The NICNAME protocol is similar to the NAME/FINGER protocol (RFC 742). To access the server:

Connect to the service host (SRI-NIC)

TCP: service port 43 decimal

NCP: ICP to socket 43 decimal, establishing two 8-bit connections

Send a single "command line", ending with <CRLF>.

Receive information in response to the command line. The server closes its connections as soon as the output is finished.

EXISTING USER PROGRAMS

NICNAME has been chosen as the global name for the user program, although some sites may choose to use the more familiar name of "WHOIS". There are versions of NICNAME for Tenex, Tops-20, and Unix. The Tenex and Tops-20 programs are written in assembly language

(FAIL/MACRO), and the Unix version is written in C. They are easy to invoke, taking one argument which is passed directly to the NICNAME server at SRI-NIC. Normally it is best to use the NIC-supplied programs, if possible, since the protocol will continue to evolve. Contact NIC@SRI-NIC for copies.

COMMAND LINES AND REPLIES

A command line is normally a single name specification. The easiest way to obtain the most recent documentation on name specifications is to give the server a command line consisting of "?<CRLF>" (that is, a question-mark alone as the name specification). The response from the NICNAME server will list all possible formats that can be used.

The responses are not currently intended to be machine-readable; the information is meant to be passed back directly to a human user. The following three examples will illustrate the use of NICNAME.

Command line: ?

Response:

Please enter a name or a handle ("ident"), such as "Smith" or "SRI-NIC". Starting with a period forces a name-only search; starting with exclamation point forces handle-only. Examples:

Smith	looks for name or handle SMITH
!SRI-NIC	looks for handle SRI-NIC only
.Smith, John	looks for name JOHN SMITH only

Adding "... " to the argument will match anything from that point, e.g. "ZU..." will match ZUL, ZUM, etc.

To have the ENTIRE membership list of a group or organization, if you are asking about a group or org. shown with the record, use an asterisk character "*" directly preceding the given argument.

[CAUTION: If there are a lot of members this will take a long time!]
You may of course use exclamation point and asterisk, or a period and asterisk together.

Command line: dyer

Response:

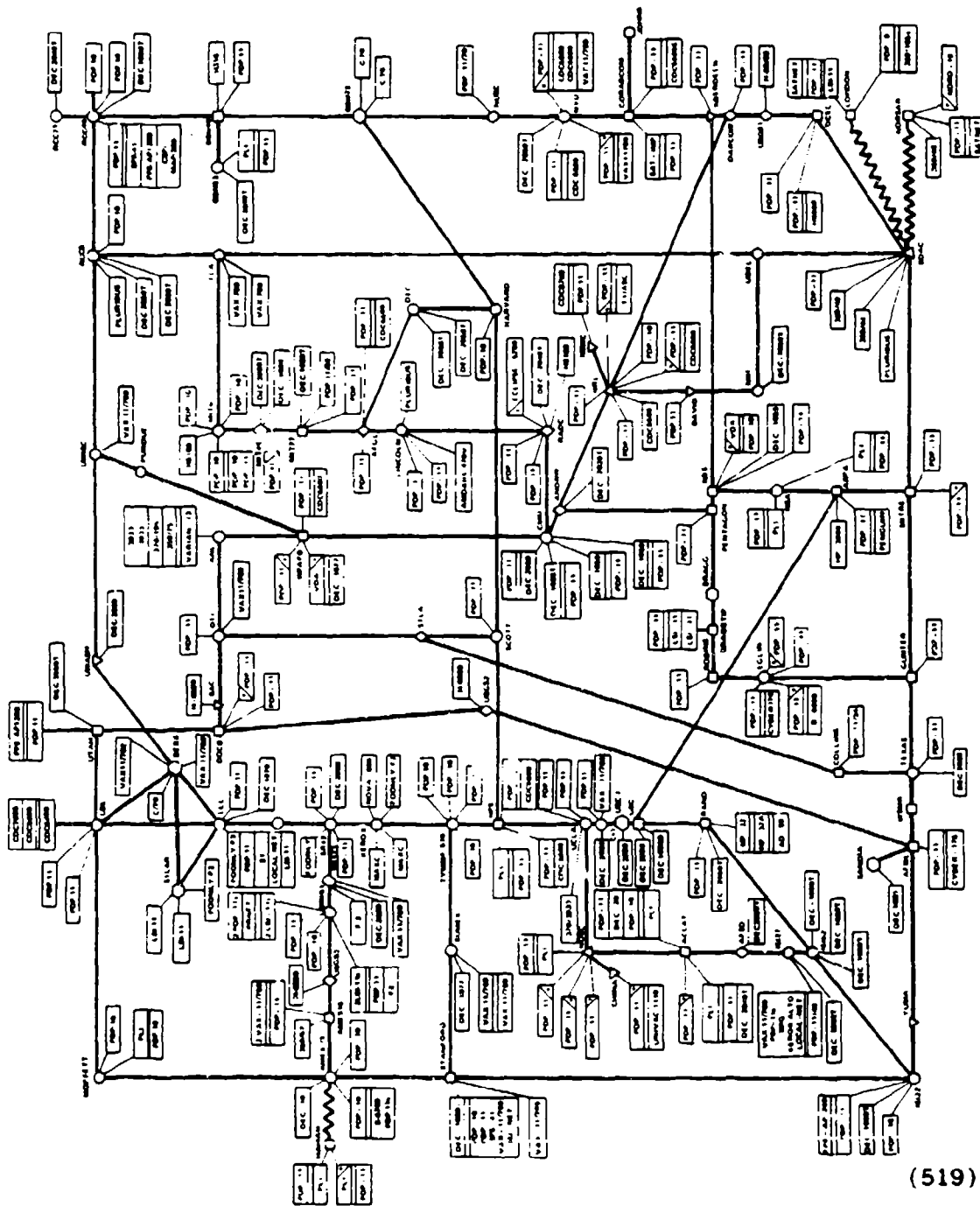
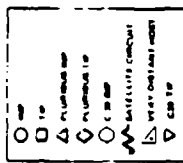
Dyer, David A. (DAD2)	DDYER@USC-ISIB	(213)	822-1511
Dyer, Fred S. (FSD)	Dyer@RADC-MULTICS	(315)	330-7275
Dyer, Mary K. (MARY)	DYER@SRI-NIC	(415)	859-4775
Dyer, William R. (WRD)	WRDyer@RADC-MULTICS	(315)	330-7791

Command line: mary

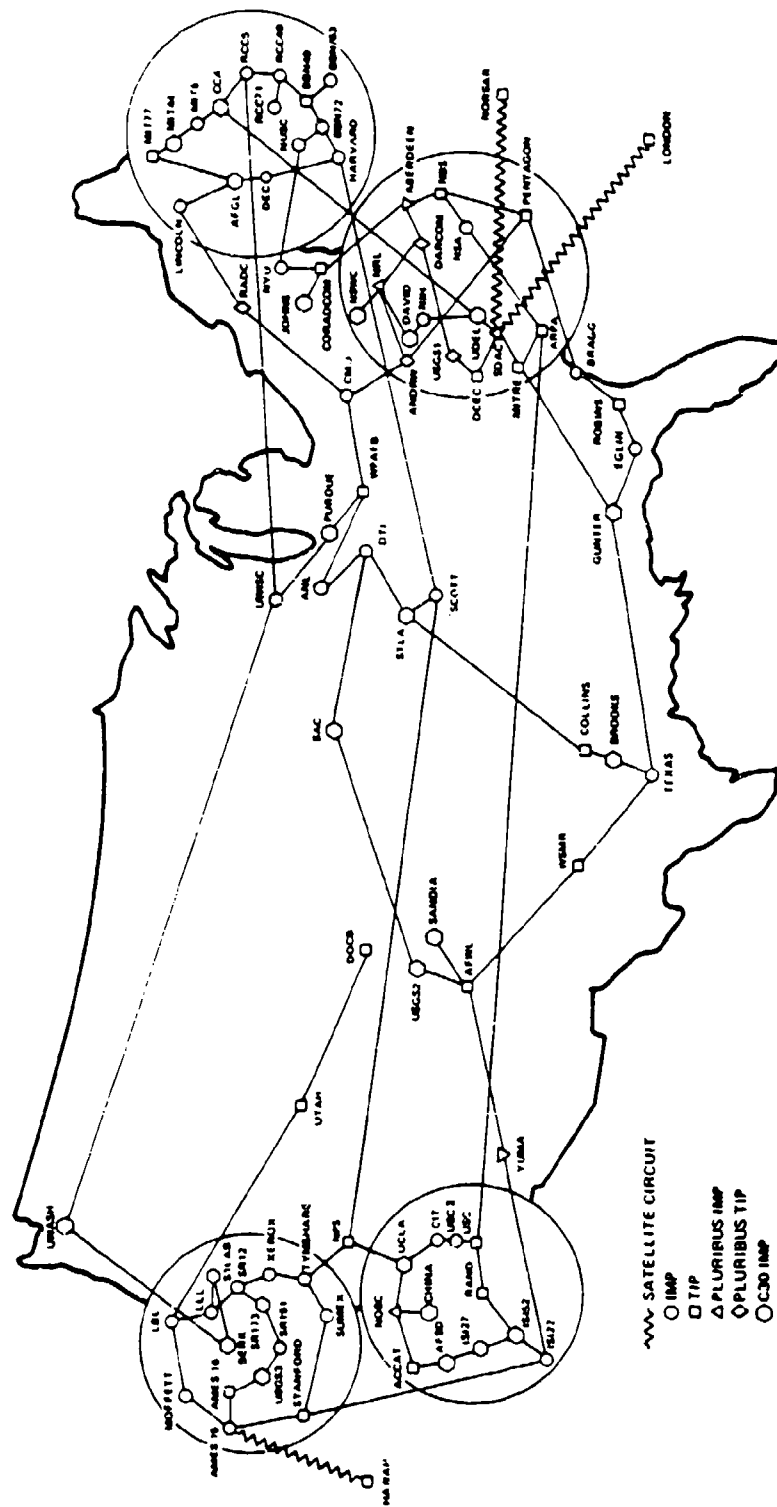
Response:

Dyer, Mary K. (MARY) DYER@SRI-NIC
SRI International
Network Information Center
Telecommunications Sciences Center
333 Ravenswood Avenue
Menlo Park, California 94025
Phone: (415) 859-4775

ARPANET LOGICAL MAP, DECEMBER 1981

[illegible]

ARPANET GEOGRAPHIC MAP, FEBRUARY 1982



- ~~~~~ SATELLITE CIRCUIT
- IMP
 - TTP
 - △ PLURIBUS IMP
 - ▽ PLURIBUS TTP
 - CSD IMP
 - ▽ CSD TTP

(NOTE: THIS MAP DOES NOT SHOW ARPANET'S EXPERIMENTAL SATELLITE CONNECTIONS)
NAMES SHOWN ARE IMP NAMES, NOT (NECESSARILY) HOST NAMES